

Canonizable Partial Order Generators and Regular Slice Languages¹

Mateus de Oliveira Oliveira

*School of Computer Science and Communication,
KTH Royal Institute of Technology, 100-44 Stockholm, Sweden
mdeoliv@kth.se*

Abstract

In a previous work we introduced slice graphs as a way to specify both infinite languages of directed acyclic graphs (DAGs) and infinite languages of partial orders. Therein we focused on the study of Hasse diagram generators, i.e., slice graphs that generate only transitive reduced DAGs. In the present work we show that any slice graph can be transitive reduced into a Hasse diagram generator representing the same set of partial orders. By employing this result we establish unknown connections between the true concurrent behavior of bounded p/t -nets and traditional approaches for representing infinite families of partial orders, such as Mazurkiewicz trace languages and Message Sequence Chart (*MSC*) languages. Going further, we identify the family of weakly saturated slice graphs. The class of partial order languages that can be represented by weakly saturated slice graphs is closed under union, intersection and even under a suitable notion of complementation (globally bounded complementation). The partial order languages in this class also admit canonical representatives in terms of Hasse diagram generators, and have decidable inclusion and emptiness of intersection. Our transitive reduction algorithm plays a fundamental role in these decidability results.

Keywords: Partial Order Languages, Regular Slice Languages,
Transitive Reduction, Petri Nets

1. Introduction

It is widely recognized that both the true concurrency and the causality between the events of concurrent systems can be adequately captured through partial orders [29, 24, 55, 38, 43]. In order to represent the whole concurrent behavior of systems, several methods of specifying infinite families of partial orders have been proposed. Partial languages [31], series-parallel languages [44], concurrent automata [19], causal automata [48], approaches derived from trace theory [23, 46, 18, 34, 42], approaches derived from message sequence chart theory [33, 25, 26], and more recently, Hasse diagram generators [16].

¹This work extends the paper [17] by the same author.

Hasse diagram generators are defined with basis on slice graphs, which by their turn, may be regarded as a specialization (modulo some convenient notational adaptations) of graph grammars [21, 13]. Indeed, slice graphs may be viewed as automata that concatenate atomic blocks called slices, to generate infinite families of directed acyclic graphs (DAGs) and to represent infinite sets of partial orders. A Hasse diagram generator \mathcal{HG} is a slice graph that generates exclusively transitive reduced graphs. In other words, every DAG in the graph language generated by \mathcal{HG} is the Hasse diagram of the partial order it represents. Such generators were introduced by us in [16] in the context of Petri net theory, and used to solve different open problems related to the partial order semantics of bounded *place/transition*-nets (*p/t*-nets). For instance, we showed that the set of partial order runs of any bounded *p/t*-net N can be represented by an effectively constructible Hasse diagram generator \mathcal{HG}_N . Previously, approaches that mapped behavioral objects to *p/t*-nets were either not expressive enough to fully capture partial order behavior of bounded *p/t*-nets, or were not guaranteed to be finite and thus, not effective [22, 47, 35, 32].

In [16] we also showed how to use Hasse diagram generators to verify the partial order behavior of concurrent systems modeled through bounded *p/t*-nets. More precisely, given a bounded *p/t*-net N with partial order behavior $\mathcal{L}_{PO}(N)$ and a HDG \mathcal{HG} representing a set $\mathcal{L}_{PO}(\mathcal{HG})$ of partial orders, we may effectively verify both whether $\mathcal{L}_{PO}(\mathcal{HG})$ is included into $\mathcal{L}_{PO}(N)$ and whether their intersection is empty. Previously an analogous verification result was only known for finite languages of partial orders [40]. As a meta-application of this verification result, we were able to test the inclusion of the partial order behavior of two bounded *p/t*-nets N_1 and N_2 : Compute \mathcal{HG}_{N_1} and test whether $\mathcal{L}_{PO}(\mathcal{HG}_{N_1}) \subseteq \mathcal{L}_{PO}(N_2)$. The possibility of performing such an inclusion test for bounded *p/t*-nets had been open for at least a decade. In the nineties, Jategaonkar-Jagadeesan and Meyer [39] proved that the inclusion of the causal behavior of 1-safe *p/t*-nets is decidable, and Montanari and Pistore [48] showed how to determine whether two bounded nets have bisimilar causal behaviors.

Finally, Hasse diagram generators may be used to address the synthesis of concurrent systems from behavioral specifications. The idea of the synthesis is appealing: Instead of constructing a system and verifying if it behaves as expected, we specify a priori which runs should be present on it, and then automatically construct a system satisfying the given specification [41, 52, 12]. In our setting the systems are modeled via *p/t*-nets and the specification is made in terms of Hasse diagram generators. In [16] we devised an algorithm that takes a Hasse diagram generator \mathcal{HG} and a bound b as input, and determines whether there is a b -bounded *p/t*-net whose partial order behavior includes $\mathcal{L}_{PO}(\mathcal{HG})$. If such a net exists, the algorithm returns the net N whose behavior minimally includes $\mathcal{L}_{PO}(\mathcal{HG})$. More precisely for every other b -bounded *p/t*-net N' satisfying $\mathcal{L}_{PO}(\mathcal{HG}) \subseteq \mathcal{L}_{PO}(N')$ it is guaranteed that $\mathcal{L}_{PO}(N) \subseteq \mathcal{L}_{PO}(N')$. This implies in particular, that if the set of runs specified by \mathcal{HG} indeed matches the partial order behavior of a b -bounded *p/t*-net N , then this net will be returned. The synthesis of *p/t*-nets from finite sets of partial orders was accomplished in [6] and subsequently generalized in [7] (see also [45]) to infinite languages specified by rational expressions over partial orders, which are nevertheless not expressive enough to represent the whole behavior of arbitrary bounded *p/t*-nets. For other results considering the synthesis of several types of Petri nets from several types of automata and languages, specifying both sequential and step behaviors we point to [20, 34, 3, 4, 14, 15].

2. Transitive Reduction of Slice Graphs and its Consequences

Both the verification and the synthesis of p/t -nets described in the previous section are stated in function of Hasse diagram generators, and do not extend directly to general slice graphs. The main goal of this paper is to overcome this limitation, by proving that any slice graph can be transitive reduced into a Hasse diagram generator specifying the same partial order language.

Theorem 2 (Transitive Reduction of General Slice Graphs). Any slice graph \mathcal{SG} can be transitive reduced into a Hasse diagram generator \mathcal{HG} representing the same partial order language, i.e., $\mathcal{L}_{PO}(\mathcal{SG}) = \mathcal{L}_{PO}(\mathcal{HG})$.

This result is interesting for two main reasons: First slice graphs are much more flexible than Hasse diagram generators from a specification point of view. Second it establishes interesting connections between p/t -nets and well known formalisms aimed to specify infinite families of partial orders, such as Mazurkiewicz trace languages [46] and message sequence chart (MSC) languages [33]. More precisely, we prove that if a partial order language \mathcal{L}_{PO} is specified through a pair (\mathcal{A}, I) of finite automaton \mathcal{A} over an alphabet of events Σ and a Mazurkiewicz independence relation $I \subseteq \Sigma \times \Sigma$, then there is a slice graph \mathcal{SG} representing the same set of partial orders. A similar result holds if \mathcal{L}_{PO} is specified by a high-level message sequence chart (HMSC), or equivalently, by a message sequence graph (MSG) [2, 51, 49]. We point out that in general, the slice graphs arising from these transformations may be far from being transitive reduced and that a direct translation of these approaches in terms of Hasse diagram generators is not evident. Nevertheless, Theorem 2 guarantees that these slice graphs can be indeed transitive reduced into Hasse diagram generators representing the same partial order language, allowing us in this way to apply both our verification and synthesis results to Mazurkiewicz trace languages and MSC languages (Corollary 3).

It is worth noting that Corollary 3 addresses the synthesis of *unlabeled* p/t -nets from partial order languages represented by traces or message sequence graphs. The synthesis of labeled p/t -nets (i.e., nets in which two transitions may be labeled by the same action) from Mazurkiewicz trace languages and from local trace languages [35] was addressed respectively in [36] and in [42]. However there is a substantial difference between labeled and unlabeled p/t -nets when it comes to partial order behavior. For instance, if we allow the synthesized nets to be labeled, we are helped by the fact that labeled 1-safe p/t -nets are already as partial order expressive as their b -bounded counterparts [8]. Thus the synthesis of unlabeled nets tends to be harder.

Our transitive reduction algorithm is also a necessary step towards the canonization of slice graphs. We say that a function \mathcal{C}_{PO} canonizes slice graphs with respect to their **partial order** languages if for every slice graph \mathcal{SG} , $\mathcal{L}_{PO}(\mathcal{SG}) = \mathcal{L}_{PO}(\mathcal{C}_{PO}(\mathcal{SG}))$ and $\mathcal{C}_{PO}(\mathcal{SG}) \sim \mathcal{C}_{PO}(\mathcal{SG}')$ for all other slice graph \mathcal{SG}' satisfying $\mathcal{L}_{PO}(\mathcal{SG}) = \mathcal{L}_{PO}(\mathcal{SG}')$. In the same way that a Hasse diagram provides a minimal representation for its induced partial order, it is natural that Hasse diagram generators correspond to the canonical forms of slice graphs. However simply transitive reducing a slice graph is not sufficient to put it into a canonical form, and indeed canonization is in general uncomputable. Fortunately,

there is a very natural and decidable² subclass of slice graphs (weakly saturated slice graphs) for which canonization is feasible. Besides admitting canonical representatives, partial order languages represented by weakly saturated slice graphs are closed under union, intersection and even under a special notion of complementation, which we call *globally bounded complementation*. Furthermore inclusion (and consequently, equality) and emptiness of intersection are decidable for this class of languages. Transitive reduction will play an important role in the definition of globally bounded complementation and, as we argue in the next paragraph, it will play a **fundamental role** in the closure, decidability and canonizability results stated above.

A slice graph \mathcal{SG} is meant to represent three distinct languages: A slice language $\mathcal{L}(\mathcal{SG})$ which is a regular subset of the free monoid generated by a slice alphabet Σ_S^c ; a graph language $\mathcal{L}_G(\mathcal{SG})$ consisting of the DAGs which have a string representative in the slice language; and a partial order language $\mathcal{L}_{PO}(\mathcal{SG})$ obtained by taking the transitive closure of DAGs in the graph language. As we will show in Section 5, any weakly saturated slice graph can be efficiently transformed into a stronger form, which we call *saturated slice graph*, representing the same graph and partial order languages. It turns out that except for complementation, operations involving languages of DAGs generated by saturated slice graphs are reflected by operations performed in their slice languages, which for being regular, have several well known decidability and computability results. This observation may be interpreted as a consequence of the fact that saturated slice languages are closed under a certain commutation operation defined on Σ_S^c . If additionally, the slice graphs in consideration are Hasse diagram generators, then questions about their partial order languages can be further mapped to questions about their graph languages, paving in this way a path to decidability. The crucial point is that this last observation fails badly if the slice graphs are not transitive reduced: There exist (**even saturated**) slice graphs \mathcal{SG} and \mathcal{SG}' for which $\mathcal{L}_G(\mathcal{SG}) \cap \mathcal{L}_G(\mathcal{SG}') = \emptyset$ but $\mathcal{L}_{PO}(\mathcal{SG}) \cap \mathcal{L}_{PO}(\mathcal{SG}') \neq \emptyset$, or for which $\mathcal{L}_G(\mathcal{SG}) \not\subseteq \mathcal{L}_G(\mathcal{SG}')$ but $\mathcal{L}_{PO}(\mathcal{SG}) \subseteq \mathcal{L}_{PO}(\mathcal{SG}')$. Thus it is essential that we transitive reduce slice graphs before performing operations with their partial order languages. With regard to this observation, an important feature of our transitive reduction algorithm is that it preserves weak saturation. The complementation of the graph and of the partial order languages generated by a saturated slice graphs does not follow from the closure under commutation described above, however it is still achievable in a suitable sense (globally bounded complementation), whose definition we postpone to Section 5.

A skeptic could wonder whether weak saturation is an excessively strong condition which could be only satisfied by uninteresting examples of slice graphs. We counter this skepticism by describing three natural situations in which weakly saturated slice graphs arise: The first two examples stem from the fact that our study of weakly saturated slice languages was inspired, and indeed generalizes, both the theory of recognizable trace languages [46] and the theory of linearization-regular³ message sequence languages [33]. In particular, recognizable trace languages can be mapped to weakly saturated regular slice

²In [33] it is undecidable whether a MSC-language is linearization-regular. This is not in contradiction with the decidability of weak saturation. An analogous statement for us would be: It is undecidable whether a slice graph can be weakly saturated.

³In our work the term regular is used in the standard sense of finite automata theory. The notion of "regular" used in [33] is analogous to our notion of regular+saturated.

languages, while linearization-regular MSC languages which are representable by message sequence graphs, may be mapped to *loop connected* slice graphs, which can be efficiently weakly saturated. Our third and most important example comes from the theory of bounded p/t -nets. More precisely, we show that the Hasse diagram generators associated to bounded p/t -nets in [16] are saturated. This last observation has two important consequences: first, slice graphs are strictly more expressive than both Mazurkiewicz trace languages, and MSC-languages, since there exist even 1-safe p/t -nets whose partial order behavior cannot be expressed through these formalisms; second, it implies that the behavior of bounded p/t -nets may be canonically represented by Hasse diagram generators. While in [16] we were able to associate a HDG \mathcal{HG}_N to any bounded p/t -net N , we were not able to prove that if two nets N and N' have the same partial order behavior then they can be associated to same HDG⁴. By showing that the partial order language of bounded p/t -nets may be represented via saturated slice languages we are able to achieve precisely this goal (Theorem 8):

Theorem 8 (Intuitive Version) . The set $\mathcal{L}_{PO}(N)$ of partial order runs of any bounded p/t -net N can be canonically represented by a saturated Hasse diagram generator $\mathcal{HG}(N)$. In particular for any other bounded p/t -net N' such that $\mathcal{L}_{PO}(N) = \mathcal{L}_{PO}(N')$ it holds that $\mathcal{HG}(N) = \mathcal{HG}(N')$.

The rest of the paper is organized as follows: Next, in Section 3 we define slices, slice graphs and slice languages. Subsequently, in sections 4 and 5 we introduce the main contributions of this work, which are our transitive reduction algorithm (Section 4) and our study of partial order languages that can be represented through saturated slice languages (Section 5). In section 6 we prove that both Mazurkiewicz trace languages and MSC-languages can be mapped to slice languages. In section 7 we show how our results may be used as a link between Mazurkiewicz traces, MSC languages and the partial order behavior of p/t -nets. Finally in Section 8 we make some final comments.

3. Slices

There are several automata-theoretic approaches for the specification of infinite families of graphs: graph automata [54, 11], automata over planar DAGs [9], graph rewriting systems [13, 5, 21], and others [28, 27, 10]. In this section we will introduce an approach that is more suitable for our needs. Namely, the representation of infinite families of DAGs with bounded slice width. In particular, the slices defined in this section can be regarded as a specialized version of the multi-pointed graphs defined in [21], which are too general, and which are subject to a slightly different notion of concatenation.

A *slice* is a labeled DAG $\mathbf{S} = (V, E, l)$ whose vertex set V is partitioned into three subsets: A non-empty center C labeled by l with the elements of an arbitrary set T of events, and the in- and out-frontiers I and O respectively which are numbered by l in such a way that $l(I) = \{1, \dots, |I|\}$ and $l(O) = \{1, \dots, |O|\}$. Furthermore a unique edge in E touches each frontier vertex $v \in I \dot{\cup} O$, where $\dot{\cup}$ denotes the disjoint union of sets. This

⁴In general, a partial order language can be represented by several distinct Hasse diagram generators.

edge is outgoing if v lies on the in-frontier I and incoming if v lies on the out-frontier O . In drawings, we surround slices by dashed rectangles, and implicitly direct their edges from left to right. In-frontier and out-frontier vertices are determined respectively by the intersection of edges with the left and right sides of the rectangle. Frontier vertices are implicitly numbered from top to bottom. Center vertices are indicated by their labels (Fig. 1-*i*).

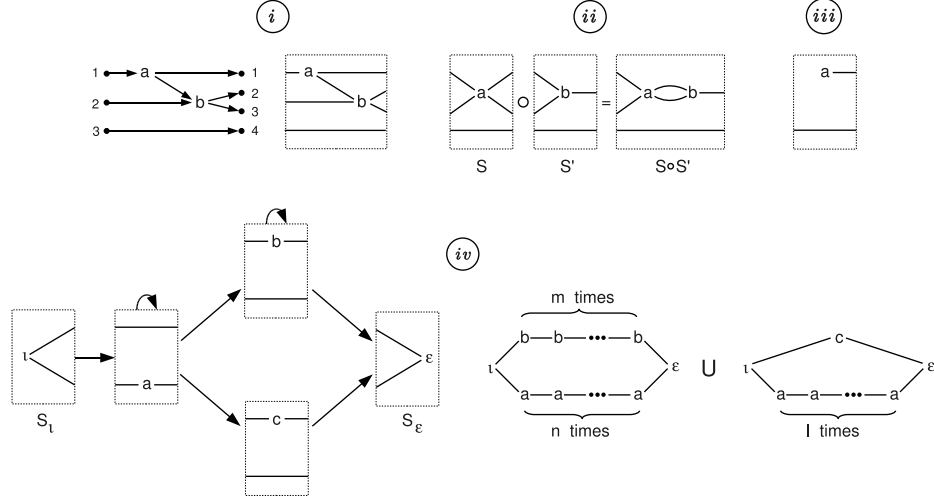


Figure 1: *i*) A slice *ii*) Composition of slices. *iii*) A degenerate slice. *iv*) A slice graph labeled with unit slices, and an intuitive representation of its graph language. S_l is initial and S_ε , final.

A slice S_1 can be composed with a slice S_2 whenever the out-frontier of S_1 is of the same size as the in-frontier of S_2 . In this case, the resulting slice $S_1 \circ S_2$ is obtained by gluing the single edge touching the j -th out-frontier vertex of S_1 to the corresponding edge touching the j -th in-frontier vertex of S_2 (Fig. 1-*ii*). We note that as a result of the composition, multiple edges may arise, since the vertices on the glued frontiers disappear. A slice with a unique vertex in the center is called a *unit slice*. A sequence of unit slices $S_1 S_2 \dots S_n$ is a *unit decomposition* of a slice S if $S = S_1 \circ S_2 \circ \dots \circ S_n$. The definition of unit decomposition extends to DAGs by regarding them as slices with empty in and out-frontiers. The slice-width of a slice is defined as the size of its greatest frontier. The slice width of a unit decomposition $S = S_1 \circ S_2 \circ \dots \circ S_n$ is the slice-width of its widest slice. The *existential slice-width* of a DAG G is the slice width of its thinnest unit decomposition. The *global slice-width* of a DAG G is the width of its widest unit decomposition.

We say that a slice is *initial* if its in-frontier is empty and *final* if its out-frontier is empty. A unit slice is non-degenerate if its center vertex is connected to at least one in-frontier (out-frontier) vertex whenever the in-frontier (out)-frontier is not empty. In Fig. 1-*iii* we depict a degenerate unit slice. A *slice alphabet* is any finite set Σ_S of slices. The slice alphabet of width c over a set of events T is the set Σ_S^c of all unit slices of width at most c , whose center vertex is labeled with an event from T . A *slice language* over a slice alphabet Σ_S is a subset $\mathcal{L} \subseteq \Sigma_S^*$ where for each string $S_1 S_2 \dots S_n \in \mathcal{L}$, S_1 is initial, S_n is final and S_i can be composed with S_{i+1} for $1 \leq i < n$. From a slice language \mathcal{L}

we may derive a language \mathcal{L}_G of DAGs by composing the slices in the strings of \mathcal{L} , and a language \mathcal{L}_{PO} of partial orders, by taking the transitive closure of each DAG in \mathcal{L}_G :

$$\mathcal{L}_G = \{\mathbf{S}_1 \circ \mathbf{S}_2 \circ \dots \circ \mathbf{S}_n \mid \mathbf{S}_1 \mathbf{S}_2 \dots \mathbf{S}_n \in \mathcal{L}\} \quad \text{and} \quad \mathcal{L}_{PO} = \{H^* \mid H \in \mathcal{L}_G\} \quad (1)$$

In this paper we assume that all slices in a slice alphabet $\Sigma_{\mathbb{S}}$ are unit and non-degenerate, but this restriction is not crucial. With this assumption however, every DAG in the graph language derived from a slice language has a unique minimal and a unique maximal vertex.

A slice language is regular if it is generated by a finite automaton or by a regular expression over slices⁵. We notice that a slice language is a subset of the free monoid generated by a slice alphabet $\Sigma_{\mathbb{S}}$ and thus we do not need to make a distinction between regular and rational slice languages. In particular every slice language generated by a regular expression can be also generated by a finite automaton. Equivalently, a slice language is regular if and only if it can be generated by the slice graphs defined below [16]:

Definition 1 (Slice Graph). *A slice graph over a slice alphabet $\Sigma_{\mathbb{S}}$ is a labeled directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{S})$ possibly containing loops but without multiple edges. The function $\mathcal{S} : \mathcal{V} \rightarrow \Sigma_{\mathbb{S}}$ satisfies the following condition: $(v_1, v_2) \in \mathcal{E}$ implies that $\mathcal{S}(v_1)$ can be composed with $\mathcal{S}(v_2)$. We say that a vertex on a slice graph is initial if it is labeled with an initial slice and final if it is labeled with a final slice. We denote $\mathcal{L}(\mathcal{G})$ the slice language generated by \mathcal{G} , which we define as:*

$$\mathcal{L}(\mathcal{G}) = \{\mathcal{S}(v_1)\mathcal{S}(v_2)\dots\mathcal{S}(v_n) : v_1v_2\dots v_n \text{ is a walk on } \mathcal{G} \text{ from an initial to a final vertex}\}$$

We write respectively $\mathcal{L}_G(\mathcal{G})$ and $\mathcal{L}_{PO}(\mathcal{G})$ for the graph and the partial order languages derived from $\mathcal{L}(\mathcal{G})$. A slice language \mathcal{L} is *transitive reduced* if all DAGs in \mathcal{L}_G are simple and transitive reduced. In other words, each DAG in \mathcal{L}_G is the Hasse diagram of a partial order in \mathcal{L}_{PO} . A slice graph is a *Hasse diagram generator* if its slice language is transitive reduced.

4. Sliced Transitive Reduction

In [16] we devised a method to filter out from the graph language of a slice graph \mathcal{G} all DAGs which are not transitive reduced. In this way we were able to obtain a Hasse diagram generator \mathcal{HG} whose graph language consists precisely on the Hasse diagrams generated by \mathcal{G} (i.e. $\mathcal{L}_{PO}(\mathcal{HG}) \subseteq \mathcal{L}_{PO}(\mathcal{G})$). The method we devised therein falls short of being a transitive reduction algorithm, since the partial order generated by the resulting slice graph \mathcal{HG} could be significantly shrunk and indeed even reduced to the empty set. It was not even clear whether such a task could be accomplished at all, since we are dealing with applying a non-trivial algorithm, i.e. the transitive reduction, to an infinite number

⁵The operation of the monoid is just the concatenation $\mathbf{S}_1\mathbf{S}_2$ of slice symbols \mathbf{S}_1 and \mathbf{S}_2 and should not be confused with the composition $\mathbf{S}_1 \circ \mathbf{S}_2$ of slices.

of DAGs at the same time. Fortunately in this section we prove that such a transitive reduction is accomplishable, by developing an algorithm that takes a slice graph as input and returns a Hasse diagram generator \mathcal{HG} satisfying $\mathcal{L}_{PO}(\mathcal{SG}) = \mathcal{L}_{PO}(\mathcal{HG})$.

The difficulty in devising an algorithm to transitive reduce slice graphs stems from the fact that a slice that labels a vertex v of a slice graph may be used to form both DAGs which are transitive reduced and DAGs which are not, depending on which path we are considering in the slice graph. This observation is illustrated in Figure 3.ii where the slice containing the event a has this property. Thus in general the transitive reduction cannot be performed independently on each slice of the slice graph. To overcome this difficulty we will introduce in Definition 2 and in Lemma 1 a "sliced" characterization of superfluous edges of DAGs, i.e., edges that do not carry any useful transitivity information. By expanding each slice of the slice graph with a set of specially tagged copies satisfying the conditions listed in Definition 2 and connecting them in a special way, we will be able to keep all paths which give rise to transitive reduced DAGs, and to create new paths which will give rise to transitive reduced versions of the non-transitive DAGs generated by the original slice graph. In the proof of Theorem 1 we develop an algorithm that transitive reduces slice graphs which do not generate DAGs with multiple edges. Subsequently, in Theorem 2 we will eliminate the restriction on multiple edges and prove that slice graphs in general can be transitive reduced.

We say that an edge e of a simple DAG H is superfluous if the transitive closure of H equals the transitive closure of $H \setminus \{e\}$. In this section we will develop a method to highlight the sliced parts of superfluous edges of a graph H on any of its unit decompositions $\mathbf{S}_1 \mathbf{S}_2 \cdots \mathbf{S}_n$ (Fig. 2-*vi*). Deleting these highlighted edges from each slice of the decomposition, we are left with a unit decomposition $\mathbf{S}'_1 \mathbf{S}'_2 \cdots \mathbf{S}'_n$ of the transitive reduction of H . It turns out that we may transpose this process to slice graphs. Thus given a slice graph \mathcal{SG} we will be able to effectively compute a Hasse diagram generator \mathcal{HG} that represents the same language of partial order as \mathcal{SG} , i.e. $\mathcal{L}_{PO}(\mathcal{SG}) = \mathcal{L}_{PO}(\mathcal{HG})$.

A function $\mathcal{T} : E^2 \rightarrow \{0, 1\}^2$ defined on the edges of a unit slice $\mathbf{S} = (\{v\}, E, l)$ is called a coloring of \mathbf{S} . A sequence of functions $\mathcal{T}_1 \mathcal{T}_2 \cdots \mathcal{T}_n$ is a coloring of a unit decomposition $\mathbf{S}_1 \mathbf{S}_2 \cdots \mathbf{S}_n$ of a DAG H if each \mathcal{T}_i is a coloring of \mathbf{S}_i and if the colors associated by \mathcal{T}_i to pairs of edges touching the out-frontier of \mathbf{S}_i agree with the colors associated by \mathcal{T}_{i+1} to pairs of edges touching the in-frontier of \mathbf{S}_{i+1} (Figs. 2-*i*, 2-*ii*).



Figure 2: The transitivity coloring of the unit decomposition of two DAGs. *i*) The DAG is transitive reduced. No edge is marked. *ii*) The DAG is not transitive reduced. The sliced parts of each superfluous edge are marked (in gray). Deleting the marked edges and composing the slices we are left with the transitive reduction of the original DAG.

Below we define the notion of *transitivity coloring* that will allow us to perform a "sliced" transitive reduction on DAGs. We say that an edge e of a slice \mathbf{S} is marked by \mathcal{T} if $\mathcal{T}(ee) = 11$ and unmarked if $\mathcal{T}(ee) = 00$.

Definition 2 (Transitivity Coloring). *Let $\mathbf{S} = (I \cup \{v\} \cup O, E, l)$ be a unit slice. Then a transitivity coloring of \mathbf{S} is a partial function $\mathcal{T} : E^2 \rightarrow \{0, 1\}^2$ such that*

1. *Undefinedness: $\mathcal{T}(e_1 e_2)$ is not defined if and only if $(e_1^s = e_2^t)$ or $(e_1^t = e_2^s)$*
2. *Antisymmetry: If $\mathcal{T}(e_1 e_2) = ab$ then $\mathcal{T}(e_2 e_1) = ba$.*
3. *Marking: $\mathcal{T}(ee) \in \{00, 11\}$. e is unmarked if $\mathcal{T}(ee) = 00$ and marked if $\mathcal{T}(ee) = 11$.*
4. *Transitivity:*
 - (a) *If e_1 and $e_2 \neq e_1$ have the same source vertex, then $\mathcal{T}(e_1 e_2) = 00$.*
 - (b) *If $e_1^s \in I$ and $e_1^t \in O$ and $e_2^s = v$ then $\mathcal{T}(e_1 e_2) \in \{01, 11\}$ and $\mathcal{T}(e_1 e_2) = 01$ iff $(\exists e, e^t = v)(\mathcal{T}(e_1 e) \in \{00, 01\})$*
5. *Relationship between marking and transitivity:*

$$\text{If } e^t = v \text{ then } e \text{ is marked} \Leftrightarrow (\exists e_1, e_1^t = v) \mathcal{T}(ee_1) = 01$$

We observe that an isolated unit slice may be transitivity colored in many ways. However as stated in the next lemma (Lemma 1), a unit decomposition $\mathbf{S}_1 \mathbf{S}_2 \cdots \mathbf{S}_n$ of a simple DAG H with a unique minimal and a unique maximal vertices, can be coherently colored in a unique way. Furthermore, in this unique coloring, each superfluous edge of H is marked. Later, in Lemma 2 we will provide a generalization of Lemma 1 that takes DAGs with multiple edges into consideration.

Lemma 1 (Sliced Transitive Reduction). *Let $\mathbf{S}_1 \mathbf{S}_2 \cdots \mathbf{S}_n$ be a unit decomposition of a simple DAG H with a unique minimal and a unique maximal vertices. Then*

1. *$\mathbf{S}_1 \mathbf{S}_2 \cdots \mathbf{S}_n$ has a unique transitivity coloring $\mathcal{T}_1 \mathcal{T}_2 \cdots \mathcal{T}_n$.*
2. *An edge e in \mathbf{S}_i is marked by \mathcal{T}_i if and only if e is a sliced part of a superfluous edge of H (Fig. 2-ii).*

Proof. Let $\mathcal{T}_1 \mathcal{T}_2 \cdots \mathcal{T}_n$ be a transitivity coloring of $\mathbf{S}_1 \mathbf{S}_2 \cdots \mathbf{S}_n$. By the rule of composition of colored slices and by conditions 1 to 5 of Definition 2, the value associated by \mathcal{T}_i to each two distinct edges of \mathbf{S}_i are completely determined by the values associated by \mathcal{T}_{i-1} to edges touching the out-frontier of \mathbf{S}_{i-1} . Furthermore, since H has a unique minimal vertex, \mathcal{T}_1 associates the value 00 to each two distinct edges of \mathbf{S}_1 . Thus the values associated by each \mathcal{T}_i to distinct edges of \mathbf{S}_i are unique. It remains to show that the marking is unique.

Let e be a superfluous edge of H , and $e_1 e_2 \dots e_k$ be a path from e^s to e^t , then the transitivity conditions in Definition 2.4 assure that for any sliced part e' of e and any sliced part e'_i of e_i lying in the same slice \mathbf{S}_j , $\mathcal{T}_j(e', e'_i) = 00$ if $i = 1$ and $\mathcal{T}_j(e', e'_i) = 01$ for $2 \leq i \leq k$ (Fig. 2). Let S_j be the slice that contains the target vertex of e , and let e' and e'_k be respectively the sliced parts of e and e_k lying in S_j . Then $\mathcal{T}(e' e'_k) = 01$ and thus, by condition 5 of Definition 2, e' is marked, implying that any sliced part of e lying in previous slices must be marked as well. Now suppose that e_1 and e_2 have the same target vertex, and that e_1 is not superfluous. Then for any sliced part e'_1 of e_1 and any sliced part e'_2 of e_2 lying in the same slice S_j , we must have $\mathcal{T}_j(e'_1, e'_2) = 10$ if e_2 is superfluous and $\mathcal{T}_j(e'_1, e'_2) = 11$ if e_2 is not superfluous. Thus by condition 5 of Definition 2, no sliced part of e_1 can be marked. We observe that $\mathcal{T}_j(e'_1, e'_2) \neq 00$ since otherwise e_1 and e_2 would have the same source and thus form a multiple edge. \square

Next, in Theorem 1 we deal with the transitive reduction of slice graphs that generate only DAGs without multiple edges, which we call *simple slice graphs*. Lemma 1 is of special importance for its proof. The transitive reduction of general slice graphs will be addressed in Theorem 2.

Theorem 1 (Transitive Reduction of Simple Slice Graphs). *Let $\mathcal{SG} = (\mathcal{V}, \mathcal{E}, \mathcal{S})$ be a slice graph such that $\mathcal{L}_G(\mathcal{SG})$ has only simple DAGs. Then there exists a Hasse diagram generator \mathcal{HG} such that $\mathcal{L}_{PO}(\mathcal{SG}) = \mathcal{L}_{PO}(\mathcal{HG})$.*

Proof. As a first step we construct an intermediary slice graph \mathcal{SG}' as follows: we expand each vertex v in \mathcal{V} with a set of vertices $\{v_{\mathcal{T}}\}$ where \mathcal{T} ranges over all transitivity colorings of $\mathcal{S}(v)$. Each vertex in $\{v_{\mathcal{T}}\}$ is labeled with $\mathcal{S}(v)$. We add an edge from $v_{\mathcal{T}}$ to $v'_{\mathcal{T}'}$ in \mathcal{SG}' if and only if v is connected to v' in \mathcal{SG} and if the values associated by \mathcal{T} to the edges touching the out-frontier of $\mathcal{S}(v)$ agree with the values associated by \mathcal{T}' to the edges touching the in-frontier of $\mathcal{S}(v')$. Finally we delete vertices that cannot be reached from an initial vertex, or that cannot reach a final vertex. We note that $\mathcal{T}_1 \mathcal{T}_2 \cdots \mathcal{T}_n$ is a transitivity coloring of the label $\mathcal{S}(v^1) \mathcal{S}(v^2) \cdots \mathcal{S}(v^n)$ of a walk from a initial vertex v_1 to a final vertex v_n in \mathcal{SG} , if and only if $\mathcal{S}(v^1) \mathcal{S}(v^2) \cdots \mathcal{S}(v^n)$ also labels the walk $v_{\mathcal{T}_1}^1 v_{\mathcal{T}_2}^2 \cdots v_{\mathcal{T}_n}^n$ in the new slice graph \mathcal{SG}' . By Lemma 1.1 a coloring exists for each such a walk and thus $\mathcal{L}_G(\mathcal{SG}) = \mathcal{L}_G(\mathcal{SG}')$. In order to get the Hasse diagram generator \mathcal{HG} with the same partial order language as \mathcal{SG} , we relabel each vertex $v_{\mathcal{T}}$ with a version of $\mathcal{S}(v)$ in which the edges which are marked by \mathcal{T} are deleted. By Lemma 1.2 a DAG is in $\mathcal{L}_G(\mathcal{HG})$ if and only if it is the transitive reduction of a DAG in \mathcal{SG} , and thus $\mathcal{L}_{PO}(\mathcal{HG}) = \mathcal{L}_{PO}(\mathcal{SG})$. \square

In general the graph language of a slice graph may contain DAGs with multiple edges. Below we extend Definition 2 to deal with these DAGs. Given a unit decomposition $\mathbf{S}_1 \mathbf{S}_2 \cdots \mathbf{S}_n$ of a DAG H , we partition each frontier of \mathbf{S}_i into numbered cells in such a way that two edges touch the same cell of a frontier if and only if they are the sliced parts of edges with the same source and target in H .

Definition 3 (Multi-edge Transitivity Coloring). *A multi-edge transitivity coloring of a unit slice $\mathbf{S} = (I \cup \{v\} \cup O, E, l)$ is a triple $(\mathcal{T}, \mathcal{P}^{in}, \mathcal{P}^{out})$ where \mathcal{T} is a transitivity coloring of \mathbf{S} , \mathcal{P}^{in} is a numbered partition of the in-frontier vertices \mathbf{S} and \mathcal{P}^{out} a numbered partition of the out-frontier vertices of \mathbf{S} , such that:*

1. *If two edges touch the same cell in one of the partitions then either both are connected to v or both touch the same cell in the other partition.*
2. *For any edge e and any distinct edges e_1 and e_2 touching the same cell of \mathcal{P}^{in} or the same cell of \mathcal{P}^{out} , $\mathcal{T}(ee_1) = \mathcal{T}(ee_2)$.*
3. *In each cell of \mathcal{P}^{in} and in each cell of \mathcal{P}^{out} either all edges are marked or all edges are unmarked.*
4. *If v is the target of two edges e_1, e_2 and $\mathcal{T}(e_1 e_2) = 00$ then e_1 and e_2 belong to the same cell of \mathcal{P}^{in} .*

A sequence $(\mathcal{T}_1, \mathcal{P}_1^{in}, \mathcal{P}_1^{out})(\mathcal{T}_2, \mathcal{P}_2^{in}, \mathcal{P}_2^{out}) \cdots (\mathcal{T}_n, \mathcal{P}_n^{in}, \mathcal{P}_n^{out})$ is a multi-edge transitivity coloring of a unit decomposition $\mathbf{S}_1 \mathbf{S}_2 \cdots \mathbf{S}_n$ of a DAG H , if $\mathcal{T}_1 \mathcal{T}_2 \cdots \mathcal{T}_n$ is a transitivity coloring of $\mathbf{S}_1 \mathbf{S}_2 \cdots \mathbf{S}_n$ and for each i , two edges e_1, e_2 in \mathbf{S}_i touch the same cell of \mathcal{P}_i^{out} if the corresponding edges to which they are glued in \mathbf{S}_{i+1} touch the same cell of \mathcal{P}_{i+1}^{in} .

3. Two edges e_1, e_2 of \mathbf{S}_i touch the same cell of \mathcal{P}_i^{in} or the same cell of \mathcal{P}_i^{out} if and only if they are sliced parts of edges e'_1, e'_2 of H with same source and tail vertices.

As a consequence our transitive reduction algorithm described in Theorem 1 may be adapted to work with general slice graphs, and not only with those that generate simple DAGs.

Theorem 2 (Transitive Reduction of General Slice Graphs). *Let $\mathcal{S}\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{S})$ be a slice graph. Then there exists a Hasse diagram generator $\mathcal{H}\mathcal{G}$ such that $\mathcal{L}_{PO}(\mathcal{S}\mathcal{G}) = \mathcal{L}_{PO}(\mathcal{H}\mathcal{G})$.*

Proof. The proof proceeds as in the proof of Theorem 1. To avoid a cumbersome notation we write \mathcal{P} for the pair $(\mathcal{P}^{in}, \mathcal{P}^{out})$. In the construction of the intermediary slice graph $\mathcal{S}\mathcal{G}'$, we expand each vertex v with a set of vertices $\{v_{\mathcal{T}, \mathcal{P}}\}$, and label each of them with $\mathcal{S}(v)$. We connect $v_{\mathcal{T}, \mathcal{P}}$ to $v'_{\mathcal{T}', \mathcal{P}'}$ if and only if v is connected to v' in $\mathcal{S}\mathcal{G}$ and both the cells of \mathcal{P} and the values of \mathcal{T} on the out-frontier of $\mathcal{S}(v)$ agree with the cells of \mathcal{P}' and the values of \mathcal{T}' on the in-frontier of $\mathcal{S}(v')$. The only point that differs in the proof is the relabeling of the vertices of $\mathcal{S}\mathcal{G}'$ in order to transform it into a Hasse diagram generator. Namely, each vertex $v_{\mathcal{T}, \mathcal{P}}$ of $\mathcal{S}\mathcal{G}'$ is relabeled with a version of $\mathcal{S}(v)$ in which for each i , the i -th cell of the partition \mathcal{P}^{in} (\mathcal{P}^{out}) is collapsed into a single in-frontier (out-frontier) vertex labeled by i , all edges touching the same cell of a partition are collapsed into a unique edge (Fig. 3.i), and all the marked edges are deleted. By Lemma 2 the DAGs generated by $\mathcal{H}\mathcal{G}$ are the transitive reduced counterparts of the DAGs generated by $\mathcal{S}\mathcal{G}$ and consequently, $\mathcal{L}_{PO}(\mathcal{H}\mathcal{G}) = \mathcal{L}_{PO}(\mathcal{S}\mathcal{G})$. \square

We end this section by giving a simple upper bound on the complexity of the transitive reduction of slice graphs:

Corollary 1. *Let $\mathcal{S}\mathcal{G}$ be a slice graph with n vertices and let s be the size of the greatest frontier of a slice labeling a vertex of $\mathcal{S}\mathcal{G}$. Then the Hasse diagram generator constructed in Theorem 2 has $n \cdot 2^{O(s^2)}$ vertices. In particular, the transitive reduction algorithm runs in polynomial time for $s = O(\sqrt{\log n})$.*

Proof. Let v be a vertex of $\mathcal{S}\mathcal{G}$ which is labeled with a slice \mathbf{S} of width s . Then there are at most $2^{O(s^2)}$ transitivity colorings of \mathbf{S} , since each two edges touching the same frontier of \mathbf{S} can be colored in at most a constant number of ways. Furthermore, there are at most $2^{O(s \log s)}$ possible ways of partitioning a set of size s , and thus of partitioning each frontier of \mathbf{S} . This implies that the number of possible multi-edge transitive colorings of \mathbf{S} is still bounded by $2^{O(s^2)}$. Since $\mathcal{S}\mathcal{G}$ has n vertices, the bound of $n \cdot 2^{O(s^2)}$ follows. \square

5. Saturated and Weakly Saturated Slice Languages

In this section we introduce *weakly saturated slice languages* and show that they allow us to smoothly generalize regular string languages to the partial order setting. In particular, the class of partial order languages which are definable through weakly saturated slice languages is closed under union intersection and even under a suitable notion of complementation, which we call *globally bounded complementation*. Furthermore, both inclusion and emptiness of intersection are decidable for this class.

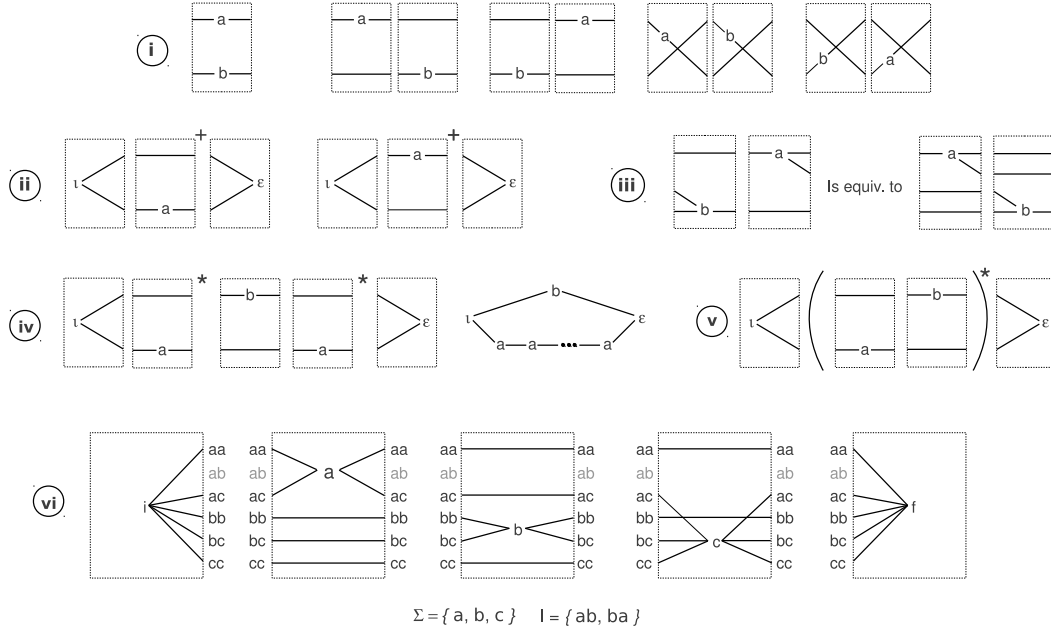


Figure 4: *i)* A slice and its set of unit decompositions. *ii)* Two regular expressions over slices generating the same graph and partial order languages, but distinct slice languages. *iii)* Equivalence of slice strings. *iv)* A slice expression over slices generating a weakly saturated slice language and an intuitive depiction of its graph language. *v)* An example of slice language that cannot be saturated. *vi)* Mapping an independence alphabet to a slice alphabet.

Weakly saturated slice languages generalize both recognizable Mazurkiewicz trace languages [46] and linearization-regular message sequence chart languages [33]. It turns out that this generalization is strict. As showed by us in [16], regular slice languages are expressive enough to represent the partial order behavior of any bounded p/t -net. Furthermore, as we will prove in Section 7, the Hasse diagram generators associated to p/t -nets in [16] are saturated. In contrast with this result, we note that the partial order behavior of bounded p/t -nets cannot be represented by Mazurkiewicz traces, which for example rule out auto-concurrency, neither by MSC languages. Indeed these formalisms are not able to capture even the partial order behavior of 1-bounded p/t -nets. On the other hand, as we will show in Section 6, both Mazurkiewicz traces and MSC languages can be reinterpreted in terms of non-transitive reduced slice languages, and through an application of our transitive reduction algorithm, they can be indeed mapped to Hasse diagram generators representing the same set of partial orders.

The following chain of implications relating the graph and partial order languages represented by two slice languages \mathcal{L} and \mathcal{L}' is a direct consequence of Equation (1):

$$\mathcal{L} \subseteq \mathcal{L}' \Rightarrow \mathcal{L}_G \subseteq \mathcal{L}'_G \Rightarrow \mathcal{L}_{PO} \subseteq \mathcal{L}'_{PO} \quad (2)$$

However there are simple examples of slice languages \mathcal{L} and \mathcal{L}' for which $\mathcal{L}_G \subseteq \mathcal{L}'_G$ and $\mathcal{L} \not\subseteq \mathcal{L}'$ (Fig. 4.ii) or for which $\mathcal{L}_{PO} \subseteq \mathcal{L}'_{PO}$ and $\mathcal{L}_G \not\subseteq \mathcal{L}'_G$ (Fig. 3). If \mathcal{L} and \mathcal{L}' are regular

slice languages, then the inclusion $\mathcal{L} \subseteq \mathcal{L}'$ can be decided by standard finite automata techniques. However, even if \mathcal{L} and \mathcal{L}' are regular slice languages it is undecidable whether $\mathcal{L}_G \subseteq \mathcal{L}'_G$ ($\mathcal{L}_{PO} \subseteq \mathcal{L}'_{PO}$) as well as whether $\mathcal{L}_G \cap \mathcal{L}'_G = \emptyset$ ($\mathcal{L}_{PO} \cap \mathcal{L}'_{PO} = \emptyset$) [16]. Indeed since slice languages strictly generalize trace languages (Section 6), these undecidability results may be regarded as an inheritance from analogous results in trace theory [37, 1]. Fortunately, these and other related problems become decidable for the weakly saturated slice languages, which we define below (Definition 4). Before, recall that a topological ordering of a DAG $H = (V, E, l)$ is an ordering $v_1 v_2 \cdots v_n$ of its vertices such that $i < j$ whenever $v_i < v_j$ in the partial ordering induced by H .

Definition 4 (Weakly Saturated Slice Graphs). *We say that a slice language \mathcal{L} is weakly saturated if for every DAG $H \in \mathcal{L}_G$ and every topological ordering $v_1 v_2 \cdots v_n$ of H , \mathcal{L} has a unit decomposition $\mathbf{S}_1 \mathbf{S}_2 \cdots \mathbf{S}_n$ of H in which v_i is the center vertex of \mathbf{S}_i , for each $1 \leq i \leq n$. A slice graph is weakly saturated if it generates a weakly saturated slice language.*

An important property of our transitive reduction algorithm (Theorem 2) is that it preserves weak saturation:

Proposition 1 (Transitive Reduction Preserves Weak Saturation). *Let \mathcal{SG} be a weakly saturated slice graph over the alphabet Σ_S^c and \mathcal{HG} be the transitive reduced version of \mathcal{SG} after the application of Theorem 2. Then \mathcal{HG} is weakly saturated.*

Proof. The proof follows from the fact that an ordering v_1, v_2, \dots, v_n of the vertices of a DAG G on n vertices is a topological ordering of G if and only if it is also a topological ordering of the Hasse diagram of G . \square

Even though *weak saturation* is the concept that is meant to be used in practice, in proofs it will be more convenient to deal with the notion of *saturation*, which we define below (Definition 5). In the end of this section (Theorem 5) we will show that each weakly saturated slice graph can be efficiently transformed into a saturated one generating the same partial order language, and thus all decidability results that are valid for the latter class of slice graphs are also valid for weakly saturated slice graphs. In Figure 4.iv we depict a regular expression over slices generating a weakly saturated (but not saturated) slice language.

Definition 5 (Saturated Slice Languages). *Let $ud(\mathbf{S})$ ($ud(H)$) be the set of all unit decompositions of a slice \mathbf{S} (a DAG H) (Fig. 4.i). We say that a slice language \mathcal{L} is saturated if $u(H) \subseteq \mathcal{L}$ whenever $H \in \mathcal{L}_G$. We say that a slice graph \mathcal{SG} is saturated if it generates a saturated slice language.*

It turns out that the saturation of Definition 5 can be restated in terms of the closure of a slice language, under a notion of commutation defined on its slice alphabet. Suppose that any unit decomposition of a DAG in the graph language represented by a slice language \mathcal{L} has slice width at most c . Let Σ_S^c be the set of all unit slices of slice width at most c ⁶ (Section 3). We say that two unit slices \mathbf{S} and \mathbf{S}' in Σ_S^c are independent of each

⁶More precisely $\Sigma_S^c(T)$ for some set of events T .

other if there is no edge joining the center vertex v of \mathbf{S} to the center vertex v' of \mathbf{S}' in the slice $S \circ S'$ (Fig. 4.iii). Let u and w be strings over $\Sigma_{\mathbb{S}}^c$. We say that the slice string $u\mathbf{S}_1\mathbf{S}_2w$ is similar to $u\mathbf{S}'_1\mathbf{S}'_2w$ ($u\mathbf{S}_1\mathbf{S}_2w \simeq u\mathbf{S}'_1\mathbf{S}'_2w$) if $\mathbf{S}_1 \circ \mathbf{S}_2 = \mathbf{S}'_1 \circ \mathbf{S}'_2$. The reflexive and transitive closure \simeq^* of \simeq is an equivalence class over slice strings. If the composition of the slices in a slice string $\mathbf{S}_1\mathbf{S}_2 \cdots \mathbf{S}_n$ gives rise to a DAG H , then the class of equivalence in which $\mathbf{S}_1\mathbf{S}_2 \cdots \mathbf{S}_n$ lies is equal to the set of unit decompositions of H , i.e. $ud(H)$. We observe that not necessarily every word in the free monoid generated by $\Sigma_{\mathbb{S}}^c$ corresponds to a valid graph. This however is not a problem when it comes to slice languages generated by slice graphs, and our equivalence relation on slices, gives us a way to test whether a regular slice language \mathcal{L} is saturated. As we show in the next theorem, it suffices to determine whether the minimal finite automaton generating \mathcal{L} is "diamond" closed. In Section 6.1 we will compare our notion of independence, with the notion of independence used in Mazurkiewicz trace theory.

Theorem 3. *Let \mathcal{G} be a slice graph over a slice alphabet $\Sigma_{\mathbb{S}}^c$. Then we may effectively determine whether the slice language generated by \mathcal{G} is saturated.*

Proof. Let c be the size of the largest slice labeling a vertex of \mathcal{G} . Thus the slice languages generated by \mathcal{G} is a subset of $\Sigma_{\mathbb{S}}^c$. In order to verify whether a slice graph \mathcal{G} generates a saturated slice language, it is enough to test the following condition: If a slice word $w\mathbf{S}_1\mathbf{S}_2u$ is generated by \mathcal{G} then every word $w\mathbf{S}'_1\mathbf{S}'_2u$ satisfying $\mathbf{S}'_1 \circ \mathbf{S}'_2 = \mathbf{S}_1 \circ \mathbf{S}_2$ is generated by \mathcal{G} as well. Let \mathcal{SA} be the minimal deterministic finite automaton over $\Sigma_{\mathbb{S}}$ that generates the same slice language as \mathcal{G} . Since the automaton is minimal and deterministic, any string $\mathbf{S}_1\mathbf{S}_2 \cdots \mathbf{S}_k \in \mathcal{L}(\mathcal{SA})$ corresponds to a unique computational path of \mathcal{SA} . In particular this implies that to verify our condition, we just need to determine whether \mathcal{SA} is "diamond" closed. In other words we need to test whether for each pair of transition rules $q\mathbf{S}_1r$ and $r\mathbf{S}_2q'$ of the automaton and each unit decomposition $\mathbf{S}'_1\mathbf{S}'_2$ of $\mathbf{S}_1 \circ \mathbf{S}_2$, the automaton has a state r' and transitions $q\mathbf{S}'_1r'$ and $r'\mathbf{S}'_2q'$. Clearly this condition can be effectively verified efficiently, since $\mathbf{S}_1 \circ \mathbf{S}_2$ can have at most a polynomial (on the size of $\mathbf{S}_1 \circ \mathbf{S}_2$) number of unit decompositions. \square

The class of graphs which can be represented by saturated slice languages is closed under union and intersection and has decidable inclusion and emptiness of intersection. Indeed these facts follow from the following equation, which is valid for saturated slice languages:

$$\mathcal{L} = \bigcup_{H \in \mathcal{L}_G} ud(H) \quad (3)$$

where $ud(H)$ denotes the set of all unit decompositions of H . The complement of graph languages representable by saturated slice graphs is more subtle, and **does not** follow directly from equation 4 nor from the commutation operation defined on $\Sigma_{\mathbb{S}}^c$. We say that a DAG G has *global slice width* c , if every unit decomposition of G has slice width at most c . Now suppose that a language \mathcal{L}_G of graphs can be represented by a regular and saturated slice language $\mathcal{L} \subseteq (\Sigma_{\mathbb{S}}^c)^*$. Then we define the complement of \mathcal{L}_G to be

$$\overline{\mathcal{L}_G} = \mathcal{L}_G^c \setminus \mathcal{L}_G \quad (4)$$

where \mathcal{L}_G^c is the language consisting of all DAGs of global slice width at most c . The caveat is that the fact that \mathcal{L}_G^c can be represented by a saturated regular slice language is not evident at all. Intuitively one could expect that the closure of \mathcal{L} under commutation would imply that the complement of \mathcal{L}_G could be represented at a syntactic level by intersecting $(\Sigma_S^c)^* \setminus \mathcal{L}$ with the set of all legal⁷ slice strings over Σ_S^c . As illustrated in figure 5 this intuition is misleading, and indeed $(\Sigma_S^c)^* \setminus \mathcal{L}$ may generate graphs of global slice width greater than c . The construction of a saturated slice graph \mathcal{SG}^c generating \mathcal{L}_G^c will be carried in the next subsection (Subsection 5.1), and will follow from a characterization of graphs of global slice width in terms of flows.

A similar nuance will appear when defining a suitable notion for the complementation of the partial order language \mathcal{L}_{PO} represented by a saturated slice language \mathcal{L} . We define the c -globally bounded complementation of \mathcal{L}_{PO} to be the partial order language

$$\overline{\mathcal{L}}_{PO} = \mathcal{L}_{PO}^c \setminus \mathcal{L}_{PO} \quad (5)$$

where \mathcal{L}_{PO}^c is the partial order language induced by \mathcal{L}_G^c . As we will show in the next subsection, \mathcal{L}_{PO}^c can be represented by a saturated Hasse diagram generator \mathcal{HG}^c . Four ingredients will be essential for the construction of \mathcal{HG}^c . First, the fact mentioned above that \mathcal{L}_G^c can be represented by a saturated slice graph over Σ_S^c . Second our transitive reduction algorithm (Theorem 2) which will be applied to \mathcal{SG}^c . Third, the fact that transitive reduction preserves weak saturation (Proposition 1) and finally the fact that weakly saturated slice languages can be transformed into saturated slice languages (Theorem 5).

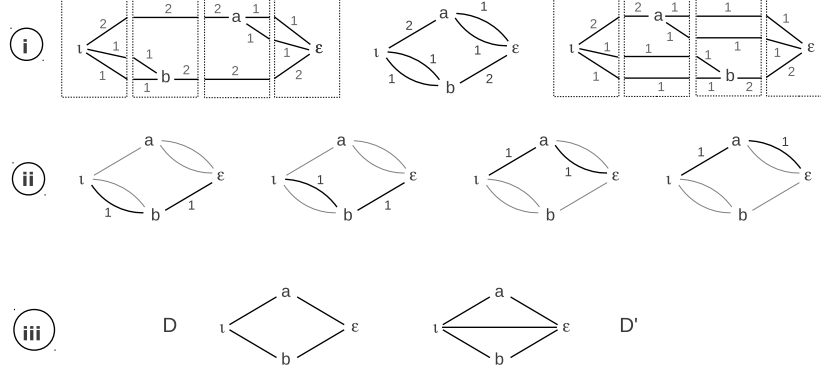


Figure 5: i) A DAG G whose edges are colored with a 4-flow f , and two unit decompositions of G colored with sliced versions of f . Suppose that \mathcal{L} is a slice language and that $G \notin \mathcal{L}_G$. Then the unit decomposition to the left belongs to $(\Sigma_S^c)^* \setminus \mathcal{L}$ for $c = 3$, but not the unit decomposition to the right, which has slice width 4. ii) A c -flow f can be regarded as the sum of c unit flows (In this case $c = 4$). iii) A diamond DAG D and a diamond with an additional edge D' . Let $\mathcal{L} = ud(D)$ and $\mathcal{L}' = ud(D')$ then both \mathcal{L} and \mathcal{L}' are saturated and $\mathcal{L}_{PO}(D) = \mathcal{L}_{PO}(D') \neq \emptyset$ but $\mathcal{L} \cap \mathcal{L}' = \emptyset$ and $\mathcal{L}_G \cap \mathcal{L}'_G = \emptyset$.

⁷By legal we mean slice strings which can be composed to form a DAG.

5.1. Globally Bounded Slice Graph and Globally Bounded Hasse Diagram Generator

In order to construct the slice graph \mathcal{S}^c (Lemma 4) representing \mathcal{L}_{PO}^c , we will need to introduce a "sliced characterization" of graphs of global slice width c . With this goal in mind, we define the notion of c -flow coloring for unit slices:

Definition 6 (c -flow coloring). *Let $\mathbf{S} = (\{v\}, E, l)$ be a unit slice. We say that a function $f : E \rightarrow \mathbb{N}$ is a c -flow coloring of \mathbf{S} if $f : E \rightarrow \mathbb{N}$ satisfies the following conditions:*

1. *Positivity: For any edge $e \in E$, $f(e) \geq 1$,*
2. *Vertex Conservativity:*
 - *If both frontiers of \mathbf{S} are non-empty, then $\sum_{e_1^s=v} f(e) = \sum_{e_2^t=v} f(e)$*
3. *Frontier Conservativity:*
 - *If the in-frontier of \mathbf{S} is non-empty then $\sum_e f(e) = c$ where e ranges over the edges touching the in-frontier of \mathbf{S} .*
 - *If the out-frontier of \mathbf{S} is non-empty then $\sum_{e'} f(e') = c$ where e' ranges over the edges touching the out-frontier of \mathbf{S} .*

A c -flow coloring of a unit decomposition $\mathbf{S}_1\mathbf{S}_2\ldots\mathbf{S}_n$ of a DAG G , is a sequence of functions $f_1f_2\ldots f_n$ such that each f_i is a c -flow coloring of \mathbf{S}_i , and such that the values associated to edges touching the out-frontier of \mathbf{S}_i agree with the values associated by f_{i+1} to edges touching the in-frontier of \mathbf{S}_{i+1} (Fig 5.i). In Lemma 3 below we assume that the DAGs have a unique minimal and a unique maximal vertex. This assumption is not at all essential and is made only for the sake of avoiding the consideration of several special cases.

Lemma 3 (c -Flows and Global Slice Width). *Let $G = (V, E, l)$ be a DAG with a unique minimal vertex v_l and a unique maximal vertex v_ε . Then G has global slice width at most c if and only if there exists a function $f : E \rightarrow \mathbb{N}$ satisfying the following conditions:*

1. *Positivity: For any edge e of G , $f(e) \geq 1$,*
2. *Vertex Conservativity: For every $v \in V$, $\sum_{e_1^s=v} f(e) = \sum_{e_2^t=v} f(e)$*
3. *Initialization and Finalization: $\sum_{e^s=v_l} f(e) = c = \sum_{e^t=v_\varepsilon} f(e)$*

Proof. Let G be a DAG and $f : E \rightarrow \mathbb{N}$ be a function satisfying conditions 1 to 3. To each slice decomposition $\mathbf{S}_1\mathbf{S}_2\ldots\mathbf{S}_n$ of G where $\mathbf{S}_i = (V_i, E_i, l_i)$, we may associate a c -flow coloring $f_1f_2\ldots f_n$ (with $f_i : E_i \rightarrow \mathbb{N}$) by setting $f_i(e) = f(e')$ whenever $e \in E_i$ is a sliced part of the edge $e' \in E$. Clearly each f_i satisfies conditions 1 and 2 of Definition 6. Condition 3 follows by induction on i . It holds for $i = 1$ by the Initialization condition of the present lemma, and holds for $i > 1$ by noticing that the sum of values associated to edges the in-frontier of a slice \mathbf{S}_i must be equal to the sum of values associated to the edges in the out-frontier of \mathbf{S}_{i-1} (Fig 5). Now since each f_i is a c -coloring of \mathbf{S}_i , by conditions 1 and 3, each frontier of \mathbf{S}_i has at most c edges, and thus G has global slice width at most c .

Now Suppose that G has global slice width c . Since G has a unique minimal vertex v_l and a unique maximal vertex v_ε , we have that G can be cast as the union of c paths (not necessarily disjoint paths) from v_l to v_ε . Let $G = \cup_w w$ be the union of these c paths from

v_i to v_ε (Fig. 5.ii). For each such a path $w = e_1e_2\dots e_{n-1}e_n \in G$ consider the function $f_w : E \rightarrow \mathbb{N}$ that associates the value 1 to each edge in w and the value 0 to each edge of G which is not in w . We claim that the function $f = \sum_{w \in W} f_w$ is a c -flow of G : It clearly satisfies Condition 1, since each edge of G belongs to at least one path. Condition 2 follows from the fact that for each intermediary vertex of the path both the edge which arrives to v and the edge that departs from v receive the value 1. Condition 3 follows from the fact that every considered path starts at v_i and finishes at v_ε . \square

\square

Corollary 2 (Sliced Characterization of Global Slice Width). *Let $G = (V, E, l)$ be a DAG with a unique minimal vertex and a unique maximal vertex. Then G has global slice width c if and only if each slice decomposition $\mathbf{S}_1\mathbf{S}_2\dots\mathbf{S}_n$ of G has a c -flow coloring $f_1f_2\dots f_n$.*

Proof. By Lemma 3, G has global slice width c if and only if G admits a c -flow f . Let $\mathbf{S}_1\mathbf{S}_2\dots\mathbf{S}_n$ be a unit decomposition of G . Then as shown in the proof of Lemma 3, if we let $f_i : E_i \rightarrow \mathbb{N}$ and set $f_i(e) = f(e')$ whenever $e \in E_i$ is a sliced part of $e \in E$, then $f_1f_2\dots f_n$ is a c -flow coloring of $\mathbf{S}_1\mathbf{S}_2\dots\mathbf{S}_n$ (Fig 5). \square

Lemma 4 (Globally Bounded Slice Graph). *For each $c \in \mathbb{N}$ with $c \geq 1$ there is a saturated slice graph \mathcal{SG}^c on $2^{O(c \log c)}$ vertices whose graph language is \mathcal{L}_G^c , i.e., the set of DAGs with global slice width at most c .*

Proof. In order to construct \mathcal{SG}^c , we create one vertex $\mathbf{v}_{\mathbf{S},f}$ for each unit slice \mathbf{S} of width at most c , and each c -flow coloring f of \mathbf{S} . We label the vertex $\mathbf{v}_{\mathbf{S},f}$ with the slice \mathbf{S} . A slice \mathbf{S} of width c has at most $2c$ edges. Since in a c -flow coloring of \mathbf{S} , each edge receives a value between 1 and c , there exist at most $c^{2c} = O(2^{O(c \log c)})$ ways of coloring \mathbf{S} . Since there are at most $2^{O(c \log c)}$ unit slices of width at most c , then \mathcal{SG}^c will still have at most $O(2^{O(c \log c)})$ vertices. Now we connect a vertex $\mathbf{v}_{\mathbf{S},f}$ to the vertex $\mathbf{v}_{\mathbf{S}',f'}$ if and only if \mathbf{S} can be glued to \mathbf{S}' and if the values associated by f to the out-frontier edges of \mathbf{S} agree with the values associated by f' to their respective edges touching the in-frontier of \mathbf{S}' . By this construction a unit decomposition $\mathbf{S}_1\mathbf{S}_2\dots\mathbf{S}_n$ of a graph G has a c -flow coloring $f_1f_2\dots f_n$ if and only if there is an accepting walk $\mathbf{v}_{\mathbf{S}_1,f_1}\mathbf{v}_{\mathbf{S}_2,f_2}\dots\mathbf{v}_{\mathbf{S}_n,f_n}$ in \mathcal{SG}^c such that $\mathbf{S}_1 \circ \mathbf{S}_2 \circ \dots \circ \mathbf{S}_n = G$. By Corollary 2, G has global slice width at most c . \square

Lemma 5 (Globally Bounded Hasse Diagram Generator). *For each $c \in \mathbb{N}$ with $c \geq 1$ there is a saturated Hasse diagram generator \mathcal{HG}^c on $2^{O(c^2)}$ vertices whose partial order language is \mathcal{L}_{PO}^c , i.e., the set of partial orders whose Hasse diagrams have global slice width at most c .*

Proof. As a first step, we construct slice graph \mathcal{SG}^c of Lemma 4 which generates precisely the set of DAGs of global slice width at most c , and has $2^{O(c \log c)}$ vertices. Subsequently, we apply our transitive reduction algorithm (Theorem 2) to obtain a Hasse diagram generator \mathcal{HG}^c on at most $2^{O(c \log c)} \cdot 2^{O(c^2)} = 2^{O(c^2)}$ vertices representing the same partial order language. By Proposition 1, \mathcal{HG}^c is weakly saturated, and thus by Theorem 5 it can be transformed into a fully saturated HDG. \square

\square

5.2. Decidability, Closures and Canonization

In this subsection we state closure, decidability and canonizability properties for the class of globally bounded DAG languages (Lemma 6) and for the class of globally bounded partial order languages that can be represented by saturated slice languages (Theorem 4).

A function \mathcal{C}_G canonizes slice graphs w.r.t. the graph language they generate if *i*) $\mathcal{L}_G(\mathcal{C}_G(\mathcal{S})) = \mathcal{L}_G(\mathcal{S})$ and *ii*) for every slice graphs $\mathcal{S}, \mathcal{S}'$, $\mathcal{C}_G(\mathcal{S})$ is isomorphic to $\mathcal{C}_G(\mathcal{S}')$ precisely when $\mathcal{L}_G(\mathcal{S}) = \mathcal{L}_G(\mathcal{S}')$. Similarly, a function \mathcal{C}_{PO} canonizes slice graphs w.r.t. the partial order language they generate if *i*) $\mathcal{L}_{PO}(\mathcal{C}_{PO}(\mathcal{S})) = \mathcal{L}_{PO}(\mathcal{S})$ and *ii*) for every slice graphs $\mathcal{S}, \mathcal{S}'$, $\mathcal{C}_{PO}(\mathcal{S})$ is isomorphic to $\mathcal{C}_{PO}(\mathcal{S}')$ precisely when $\mathcal{L}_{PO}(\mathcal{S}) = \mathcal{L}_{PO}(\mathcal{S}')$. We notice that it is hopeless to try to devise a canonization algorithm that works for every slice graph both with respect to their graph languages and with respect to their partial order languages. For instance, if we were able to compute canonical forms for graph languages $\mathcal{L}_G, \mathcal{L}'_G$ represented by general slice graphs, we would be able to decide $\mathcal{L}_G \subseteq \mathcal{L}'_G$ by testing whether $\mathcal{L}_G \cup \mathcal{L}'_G = \mathcal{L}'_G$ ⁸. However, inclusion of the graph languages generated by slice graphs is known to be undecidable [16]. Fortunately, as stated in Lemma 6 and in Theorem 4, such canonizability results are accomplishable for the class of saturated slice graphs.

Lemma 6 (DAG languages: Computability, Decidability and Canonization). *Let \mathcal{S} and \mathcal{S}' be two slice graphs over the alphabet Σ_S^c generating slice languages \mathcal{L} and \mathcal{L}' respectively, and suppose \mathcal{S} is saturated. Then*

1. *one may compute*
 - *a slice graph \mathcal{S}_G^\cup whose graph language is $\mathcal{L}_G \cup \mathcal{L}'_G$,*
 - *a slice graph \mathcal{S}_G^\cap whose graph language is $\mathcal{L}_G \cap \mathcal{L}'_G$ and,*
 - *a slice graph $\overline{\mathcal{S}}_G$ whose graph language is $\overline{\mathcal{L}_G} \cap \mathcal{L}_G^c$.*
- furthermore, if \mathcal{S}' is also saturated then so are $\mathcal{S}_G^\cup, \mathcal{S}_G^\cap$.*
2. *one may decide*
 - *whether $\mathcal{L}'_G \subseteq \mathcal{L}_G$ and,*
 - *whether $\mathcal{L}_G \cap \mathcal{L}'_G = \emptyset$.*
3. *one may compute a canonical saturated slice graph $\mathcal{C}_G(\mathcal{S})$ generating \mathcal{L}_G .*

Proof. Since \mathcal{L} is saturated, equation 3 implies that $\mathcal{L}_G \cup \mathcal{L}'_G$ iff $\mathcal{L} \cup \mathcal{L}'$, $\mathcal{L}_G \cap \mathcal{L}'_G$ iff $\mathcal{L} \cap \mathcal{L}'$, $\mathcal{L}'_G \subseteq \mathcal{L}_G$ iff $\mathcal{L}' \subseteq \mathcal{L}$ and $\mathcal{L}_G \cap \mathcal{L}'_G = \emptyset$ iff $\mathcal{L} \cap \mathcal{L}' = \emptyset$, while $\overline{\mathcal{S}}$ is the slice graph whose slice language is $\mathcal{L}(\mathcal{S}^c) \setminus \mathcal{L}$, where \mathcal{S}^c is the slice graph constructed in Lemma 4. Since it is well known that regular languages are closed under union, intersection and complementation, and since $\mathcal{L}, \mathcal{L}'$ and $\mathcal{L}(\mathcal{S}^c)$ are regular subsets of $(\Sigma_S^c)^*$, items 1 and 2 follow. Also, regular language theory says that there is a minimal canonical deterministic finite automaton \mathcal{A} over Σ_S^c generating \mathcal{L} . By fixing a function h that maps automata to labeled graphs representing the same regular language (e.g. see Appendix of [16]), we may set the canonical form $\mathcal{C}(\mathcal{S})$ to be the slice graph $h(\mathcal{A})$. Since \mathcal{L} is saturated, $h(\mathcal{A})$ will also be a canonical representative for the graph language \mathcal{L}_G . \square

⁸Clearly $\mathcal{L}'' = \mathcal{L} \cup \mathcal{L}'$ if and only if $\mathcal{L}''_G = \mathcal{L}_G \cup \mathcal{L}'_G$.

As noted in Section 2, there exist (**even saturated**) regular slice languages \mathcal{L} and \mathcal{L}' for which $\mathcal{L}_G \cap \mathcal{L}'_G = \emptyset$ but $\mathcal{L}_{PO} \cap \mathcal{L}'_{PO} \neq \emptyset$, or for which $\mathcal{L}_G \not\subseteq \mathcal{L}'_G$ but $\mathcal{L}_{PO} \subseteq \mathcal{L}'_{PO}$. For instance, consider the "diamond" graph D of Figure 5.iii, and a graph D' obtained from D by adding an edge from its minimal to its maximal vertex. Then the languages $\mathcal{L} = ud(D)$ and $\mathcal{L}' = ud(D')$ consisting of all unit decompositions of D and D' respectively, are saturated slice languages. However both $\mathcal{L} \cap \mathcal{L}' = \emptyset$ and $\mathcal{L}_G \cap \mathcal{L}'_G = \emptyset$, while $\mathcal{L}_{PO} \cap \mathcal{L}'_{PO}$ is not empty, since D and D' induce the same partial order. Thus, as it will be clear in the proof of the next theorem (Theorem 4), our transitive reduction algorithm is essential for the statement of decidability and computability results concerning the partial order languages represented by slice graphs.

Theorem 4 (Partial Order Languages: Computability, Decidability and Canonization). *Let $\mathcal{S}\mathcal{G}$ and $\mathcal{S}\mathcal{G}'$ be two slice graphs over the alphabet Σ_S^c generating slice languages \mathcal{L} and \mathcal{L}' respectively, and suppose $\mathcal{S}\mathcal{G}$ is saturated. Then*

1. *one may compute*
 - *a slice graph $\mathcal{S}\mathcal{G}_{PO}^{\cup}$ whose partial order language is $\mathcal{L}_{PO} \cup \mathcal{L}'_{PO}$,*
 - *a slice graph $\mathcal{S}\mathcal{G}_{PO}^{\cap}$ whose partial order language is $\mathcal{L}_{PO} \cap \mathcal{L}'_{PO}$ and,*
 - *a saturated slice graph $\overline{\mathcal{S}\mathcal{G}}_{PO}$ whose partial order language is $\overline{\mathcal{L}}_{PO} \cap \mathcal{L}_{PO}^c$.*
- furthermore, if $\mathcal{S}\mathcal{G}'$ is also saturated then so are $\mathcal{S}\mathcal{G}_{PO}^{\cup}$ and $\mathcal{S}\mathcal{G}_{PO}^{\cap}$.*
2. *one may decide*
 - *whether $\mathcal{L}_{PO}(\mathcal{S}\mathcal{G}') \subseteq \mathcal{L}_{PO}(\mathcal{S}\mathcal{G})$ and,*
 - *whether $\mathcal{L}_{PO}(\mathcal{S}\mathcal{G}) \cap \mathcal{L}_{PO}(\mathcal{S}\mathcal{G}') = \emptyset$.*
3. *one may compute a canonical saturated Hasse diagram generator $\mathcal{C}_{PO}(\mathcal{S}\mathcal{G})$ generating \mathcal{L}_{PO} .*

Proof. As a crucial step towards all the results stated in the present theorem, we apply our transitive reduction algorithm to both $\mathcal{S}\mathcal{G}$ and $\mathcal{S}\mathcal{G}'$ (Theorem 2), obtaining in this way a Hasse diagram generator $\mathcal{H}\mathcal{G}$ and $\mathcal{H}\mathcal{G}'$ representing the same partial order languages as $\mathcal{S}\mathcal{G}$ and $\mathcal{S}\mathcal{G}'$ respectively. The cruciality of this step stems from the fact that several (**even saturated**) slice graphs may represent the same partial order language. Proposition 1 guarantees that $\mathcal{H}\mathcal{G}$ is weak saturated, and thus it can be transformed into a fully saturated HDG by Theorem 5. Since the graph languages \mathcal{L}_G and \mathcal{L}'_G generated by $\mathcal{H}\mathcal{G}$ and $\mathcal{H}\mathcal{G}'$ respectively are transitive reduced, their partial order languages are in a bijective correspondence with their respective graph languages, and thus $\mathcal{L}_{PO} \cup \mathcal{L}'_{PO}$ iff $\mathcal{L}_G \cup \mathcal{L}'_G$, $\mathcal{L}_{PO} \cap \mathcal{L}'_{PO}$ iff $\mathcal{L}_G \cap \mathcal{L}'_G$, $\mathcal{L}'_{PO} \subseteq \mathcal{L}_{PO}$ iff $\mathcal{L}'_G \subseteq \mathcal{L}_G$, and $\mathcal{L}_{PO} \cap \mathcal{L}'_{PO} = \emptyset$ iff $\mathcal{L}_G \cap \mathcal{L}'_G = \emptyset$. Thus $\mathcal{S}\mathcal{G}_{PO}^{\cup} = \mathcal{H}\mathcal{G}_G^{\cup}$, $\mathcal{S}\mathcal{G}_{PO}^{\cap} = \mathcal{H}\mathcal{G}_G^{\cap}$ and the canonical form $\mathcal{C}_{PO}(\mathcal{S}\mathcal{G}) = \mathcal{C}_G(\mathcal{H}\mathcal{G})$ can be computed by using Lemma 6. Similarly inclusion and emptiness of intersection can be decided by applying Lemma 6. In order to compute $\overline{\mathcal{S}\mathcal{G}}_{PO}$, instead of applying Lemma 6 we set $\overline{\mathcal{S}\mathcal{G}}_{PO}$ to be the Hasse diagram generator whose slice language is $\mathcal{L}(\mathcal{H}\mathcal{G}^c) \setminus \mathcal{L}$ where $\mathcal{H}\mathcal{G}^c$ is the Hasse diagram generator constructed in Lemma 5. \square

5.3. Weak Saturation, Saturation and Loop Connectivity

We observe that in general it is not possible to effectively transform a non-saturated slice graph \mathcal{G} into a saturated slice graph \mathcal{G}' representing the same partial order language, since this would imply canonization of arbitrary slice graphs (see Section 5.2). Indeed from results of [53] and from our view of saturated slice languages over an alphabet Σ_S^c as being closed under a commutation operation on Σ_S^c , we can conclude that even determining whether there exists a saturated slice graph \mathcal{G}' representing the same partial order language as \mathcal{G} is undecidable.

In this subsection we prove that weakly saturated slice graphs can be transformed into saturated slice graphs representing the same partial order language. From this result we conclude that all decidability results that are valid for regular saturated slice languages with regard to the partial order language they generate are equally valid for weakly saturated slice languages. We also introduce the concept of *loop-connectivity*, which is a topological property of slice graphs. Slice graphs satisfying this property can also be saturated. Both weak saturation and loop-connectivity will have applications to concurrency theory, in the sense that recognizable trace languages [36] can be mapped to weakly saturated slice languages, while linearization regular MSC languages generated by message sequence chart graphs [33] can be mapped to loop-connected slice graphs.

Theorem 5. *Let \mathcal{G} be a weakly saturated slice graph over Σ_S^c , and suppose that \mathcal{G} has n vertices. Then there exists a saturated slice graph \mathcal{G}' on $n \cdot O(2^c)$ vertices generating the same partial order language, i.e., such that $\mathcal{L}_{PO}(\mathcal{G}) = \mathcal{L}_{PO}(\mathcal{G}')$.*

Proof. We write \mathfrak{S}_n for the symmetric group on n elements. Let \mathbf{S} be a unit slice with in-frontier I and out-frontier O , π be a permutation in $\mathfrak{S}_{|I|}$ and σ be a permutation in $\mathfrak{S}_{|O|}$. We write $(\mathbf{S}, \pi, \sigma)$ for the unit slice obtained from \mathbf{S} by permuting the labels of the in-frontier vertices according to π and the labels of the out-frontier vertices according to σ . The saturated version of \mathcal{G} is obtained by replacing each vertex v in \mathcal{V} by a set of vertices $\{v_{\pi\sigma}\}$ and labeling each $v_{\pi,\sigma}$ with the slice $(\mathcal{S}(v), \pi, \sigma)$. We add an edge $(v_{\pi\sigma}, v_{\pi'\sigma'})$ to \mathcal{E}' if and only if there is an edge from v to v' in \mathcal{G} and if $\sigma = \pi'$. We note that $\mathcal{L}_G(\mathcal{G}) = \mathcal{L}_G(\mathcal{G}')$. Also, since $(\mathbf{S}, \pi, \sigma) \circ (\mathbf{S}', \pi', \sigma') = (\mathbf{S} \circ \mathbf{S}', \pi, \sigma')$ whenever $\sigma = \pi'$, we can see that each two unit decompositions $\mathbf{S}_1 \mathbf{S}_2 \cdots \mathbf{S}_n$ and $\mathbf{S}'_1 \mathbf{S}'_2 \cdots \mathbf{S}'_n$ of a DAG H corresponding to the same topological ordering $u_1 u_2 \cdots u_n$ of its vertices, are related by permutations of the labels of the frontier vertices of \mathbf{S}_i . Since \mathcal{G} is weakly saturated the slice language of the new slice graph \mathcal{G}' contains the whole set of unit decompositions $ud(H)$ of each H in $\mathcal{L}_G(\mathcal{G})$. \square

We recall that a directed graph is strongly connected if for any two vertices v and v' there is a path going from v to v' and a path from v' to v . Below we will define the notion of loop-connected slice graph. In Theorem 6 we will prove that every loop-connected slice graph can be transformed into a saturated slice graph representing the same set of partial orders.

Definition 7 (Loop-Connected Slice Graph). *A slice graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{S})$ is loop connected if for every loop $\mathbf{v}_1 \mathbf{v}_2 \dots \mathbf{v}_n \mathbf{v}_1$ in \mathcal{G} the graph obtained by gluing the out-frontier of the slice $\mathcal{S}(\mathbf{v}_1) \circ \mathcal{S}(\mathbf{v}_2) \circ \dots \circ \mathcal{S}(\mathbf{v}_n)$ with its own in-frontier has a unique strongly connected component (Fig. 7.ii).*

Theorem 6. *For every loop-connected slice graph $\mathcal{S}\mathcal{G}$ there is a saturated slice graph $\mathcal{S}\mathcal{G}'$ representing the same partial order language.*

Proof. Let $\mathcal{S}\mathcal{A}$ be the minimal deterministic automaton over the slice alphabet $\Sigma_{\mathcal{S}}$ which generates the same slice language as $\mathcal{S}\mathcal{G}$, and let n be the number of states in $\mathcal{S}\mathcal{A}$. We repeat the following procedure n times: For each path $q_1 \xrightarrow{\mathbf{S}_1} q_2 \xrightarrow{\mathbf{S}_2} q_3$ and each two slices \mathbf{S}'_1 and \mathbf{S}'_2 such that $\mathbf{S}_1 \circ \mathbf{S}_2 = \mathbf{S}'_1 \circ \mathbf{S}'_2$, add a state q'_2 to the automaton and the transitions $q_1 \xrightarrow{\mathbf{S}'_1} q'_2$ and $q'_2 \xrightarrow{\mathbf{S}'_2} q_3$ if such a state is not already present in the automaton. We claim that if $\mathcal{S}\mathcal{G}$ is loop-connected then after iterating this step n times, $\mathcal{S}\mathcal{A}$ will generate a saturated slice language representing the same set of partial orders. To see this, let $\mathcal{S}\mathcal{A}^n$ be the automaton after the n -th iteration and suppose it is not saturated. Then for some slice string $\mathbf{S}_1\mathbf{S}_2\ldots\mathbf{S}_m$ in $\mathcal{L}(\mathcal{S}\mathcal{A}^n)$ with $m > n$, there exists i such that \mathbf{S}_i is independent of \mathbf{S}_{i+1} and there are \mathbf{S}'_i and \mathbf{S}'_{i+1} such that $\mathbf{S}_i \circ \mathbf{S}_{i+1} = \mathbf{S}'_i \circ \mathbf{S}'_{i+1}$ but $\mathbf{S}_1\mathbf{S}_2\ldots\mathbf{S}'_i\mathbf{S}'_{i+1}\ldots\mathbf{S}_m$ is not in $\mathcal{L}(\mathcal{S}\mathcal{A}^n)$. This means that for some slice string $\mathbf{S}''_1\mathbf{S}''_2\ldots\mathbf{S}''_m$ in the slice language of the original slice automaton $\mathcal{S}\mathcal{A}$, and for some i, j with $j - i > n$ there is no path from the center vertex of \mathbf{S}''_i to the center vertex of \mathbf{S}''_j in the composed slice $\mathbf{S}''_1 \circ \mathbf{S}''_2 \circ \ldots \circ \mathbf{S}''_m$. From the pumping lemma for regular languages we know that there exist slice strings $x, y, z \in \Sigma_{\mathcal{S}}^*$ such that $\mathbf{S}''_i\mathbf{S}''_{i+1}\ldots\mathbf{S}''_j = xyz$ and such that $xy^rz \in \mathcal{L}(\mathcal{S}\mathcal{A})$ for every $r \geq 0$ and thus the slice string y labels a cycle in $\mathcal{S}\mathcal{A}$. Since there is no path from the center vertex of \mathbf{S}_i to the center vertex of \mathbf{S}_j then gluing the in-frontier of the slice $\mathcal{S}(y)$ with its own out-frontier, we have a graph which is not strongly connected. \square

6. Mazurkiewicz Traces, Message Sequence Chart Languages, and Slice Graphs

In this section we show how to describe two well known formalisms used in concurrency theory in terms of slices. Namely, we will show that partial order languages represented through Mazurkiewicz traces or through message-sequence-chart languages can also be represented by slice graphs. We emphasize that slice graphs that arise from natural reductions may fall short of being transitive reduced. This observation illustrates the fact that general slice graphs may be substantially easier to reason about at a preliminary stage of specification when compared with Hasse diagram generators. It also illustrates one more application of our transitive reduction algorithm: by transitive reducing these slice graphs, and applying the results of [16] we may use Mazurkiewicz traces and MSC-languages as a point of departure for the verification and synthesis of p/t -nets. This will be the topic of next section (Section 7).

6.1. Mazurkiewicz Traces

In Mazurkiewicz trace theory, partial orders are represented as equivalence classes of words over an alphabet of events [46]. Given an alphabet Σ of events and a symmetric and anti-reflexive *independence relation* $I \subseteq \Sigma \times \Sigma$, a string $\alpha ab\beta$ is defined to be similar to the string $\alpha ba\beta$ ($\alpha ab\beta \simeq \alpha ba\beta$) if aIb . A trace is then an equivalence class of the transitive and reflexive closure of \simeq^* of the relation \simeq . We denote by $[\alpha]_I$ the trace corresponding to a string $\alpha \in \Sigma^*$. A partial order $po_I(\alpha)$ is associated with a string $\alpha \in \Sigma^*$ of events in the following way: First we consider a dependence DAG $dep_I(\alpha) = (V, E, l)$ that has one vertex $v_i \in V$ labeled by the event α_i for each $i \in \{1, \dots, |\alpha|\}$. An edge connects v_i to v_j

in E if and only if $i < j$ and $(\alpha_i, \alpha_j) \notin I$. Then $po_I(\alpha)$ is the transitive closure of $dep_I(\alpha)$. One may verify that two strings induce the same partial order if and only if they belong to the same trace. The trace language induced by a string language $\mathcal{L} \subseteq \Sigma^*$ with respect to an independence relation I is the set $[\mathcal{L}]_I = \{[\alpha]_I | \alpha \in \mathcal{L}\}$ and the trace closure of \mathcal{L} is the language $\mathcal{L}^I = \cup_{\alpha \in \mathcal{L}} [\alpha]_I$. Given a finite automaton \mathcal{A} over an alphabet Σ and an independence relation $I \subset \Sigma \times \Sigma$, we denote by $\mathcal{L}(\mathcal{A})$ the regular language defined by \mathcal{A} and by $\mathcal{L}_{PO}(\mathcal{A}, I) = \{po_I(\alpha) | \alpha \in \mathcal{L}(\mathcal{A})\}$ the partial order language induced by (\mathcal{A}, I) . The next lemma (Lemma 7) says that for any finite automaton \mathcal{A} and independence relation I , there is a slice graph $\mathcal{SG}(\mathcal{A}, I)$ inducing the same partial order language as (\mathcal{A}, I) . We notice again that $\mathcal{SG}(\mathcal{A}, I)$ is not at all guaranteed to be a Hasse diagram generator.

Lemma 7 (From Traces to Slices). *Let \mathcal{A} be a finite automaton over an alphabet Σ and $I \subset \Sigma \times \Sigma$ an independence relation. Then there exists an effectively constructible slice graph $\mathcal{SG}(\mathcal{A}, I)$ such that $\mathcal{L}_{PO}(\mathcal{A}, I) = \mathcal{L}_{PO}(\mathcal{SG}(\mathcal{A}, I))$. Furthermore, if $\mathcal{L}(\mathcal{A})$ is trace closed, then \mathcal{SG} is weakly saturated.*

Proof. From an independence alphabet (Σ, I) we will derive a slice alphabet $\Sigma_{\mathcal{S}} = \{\mathbf{S}_a | a \in \Sigma\}$ (FIG. 4.vi) in such a way that the partial order $po_I(\alpha)$ induced by a string $\alpha = \alpha_1 \alpha_2 \dots \alpha_k \in \Sigma^*$ will be identical to the partial order induced by the slice string $\mathbf{S}_{\alpha_1} \mathbf{S}_{\alpha_2} \dots \mathbf{S}_{\alpha_k}$ over $\Sigma_{\mathcal{S}}$. In other words, $po_I(\alpha)$ will be equal to the transitive closure of the DAG $\mathbf{S}(\alpha) = \mathbf{S}_1 \circ \mathbf{S}_2 \circ \dots \circ \mathbf{S}_k$. We assume without loss of generality that Σ has two special symbols ι and ε that are not independent from any other symbol in Σ . The initial symbol ι appears a unique time in the beginning of each word accepted by \mathcal{A} while the final symbol ε appears a unique time at the end of each word. Let $\Sigma' = \Sigma \setminus \{\iota, \varepsilon\}$, and $D = (\Sigma' \times \Sigma') \setminus I$ be a dependence relation. For each symbol $a \in \Sigma$ we define the slice \mathbf{S}_a as follows: Both the in-frontier I and the out-frontier O of \mathbf{S}_a have $|D|$ vertices indexed by D , and the center of \mathbf{S}_a has a unique vertex v_a which is labeled by a . In symbols $I = \{I_{ab} | \{a, b\} \in D\}$ and $O = \{O_{ab} | \{a, b\} \in D\}$. For each pair $\{b, c\} \in D$ with $a \neq b$ and $a \neq c$ we add an edge (I_{bc}, O_{bc}) in \mathbf{S}_a , and for each pair $\{a, x\} \in D$ we add edges (I_{ax}, v_a) and (v_a, O_{ax}) into \mathbf{S}_a (FIG. 4.vi). We associate with ι an initial slice \mathbf{S}_{ι} , with center vertex v_{ι} and out-frontier O , and to ε , a final slice \mathbf{S}_{ε} with center vertex v_{ε} and in-frontier I . We may assume that the DAGs $\mathbf{S}(\alpha) = \mathbf{S}_{\alpha_1} \circ \mathbf{S}_{\alpha_2} \circ \dots \circ \mathbf{S}_{\alpha_k}$ and dependence DAG $dep_I(\alpha)$ associated with a string $\alpha \in \Sigma^*$ have identical sets of vertices, with vertex v_i corresponding to the i -th symbol of α . Nevertheless these DAGs are not isomorphic. Neither one is necessarily a subgraph of the other. However one can verify the following fact: for each edge $(v_i, v_j) \in dep_I(\alpha)$ there is a path in $\mathbf{S}(\alpha)$ joining vertices v_i to v_j . Conversely, for each edge (v_i, v_j) in $\mathbf{S}(\alpha)$ there is a path joining v_i to v_j in $dep_I(\alpha)$. Hence, both $\mathbf{S}(\alpha)$ and $dep_I(\alpha)$ induce the same partial order. Let $f : \Sigma \rightarrow \Sigma_{\mathcal{S}}$ be the isomorphism that maps each symbol $a \in \Sigma$ to its slice \mathbf{S}_a . Then the isomorphic image of $\mathcal{L}(\mathcal{A})$ under f is a regular⁹ slice language inducing $\mathcal{L}_{PO}(\mathcal{A}, I)$, and thus can be represented by a slice graph $\mathcal{SG}(\mathcal{A}, I)$. \square

There is a substantial difference between our notion of independence, defined on slice alphabets and the notion of independence in Mazurkiewicz trace theory. While the inde-

⁹The term *regular* here is used in a fair sense, since f maps isomorphically the free monoid generated by Σ to the free monoid generated by $\Sigma_{\mathcal{S}}$.

pendence relation on slices is determined solely with basis on the structure of the slices (Fig. 4.iii), without taking into consideration the events that label their center vertices, the Mazurkiewicz independence relation is defined directly on events. As a consequence, once an independence relation I is fixed, the nature of the partial orders that can be represented as traces with respect to I is restricted. This is valid even for more general notions of traces, such as Diekert's semi-traces [18] and the context dependent traces of [34], in which for instance, partial orders containing auto-concurrency cannot be represented. In our setting any partial order po labeled over a set of events T may be represented by a slice trace: namely the set of unit decompositions of its the Hasse diagram.

6.2. From MSC-Languages to Slice Graphs

Message Sequence Charts (MSCs) are used to depict the exchange of messages between the processes of a distributed system along a single partially ordered execution. Although being only able to represent partial orders of a very special type, MSCs find several applications and are in special suitable to describe the behavior of telecommunication protocols. Infinite families of MSCs can be specified by hierarchical (or high-level) message sequence charts (HMSCs) or equivalently, by message sequence graphs (MSGs) [2, 51, 49]. In this section we chose to work with message sequence graphs for they have a straightforward analogy with slice graphs. Namely, message sequence graphs are directed graphs without multiple edges, but possibly containing self loops, whose vertices are labeled with MSCs instead of with slices. Thus our translation from MSGs to slice graphs amounts to translate MSCs to slices in such a way that the composition of the former yields the same partial orders as the composition of the latter. We notice that the resulting slice graphs are not guaranteed to be transitive reduced. However, by using our transitive reduction algorithm, they can be further reduced in into Hasse diagram generators (Fig. 6).

We formalize MSCs according to the terminology in [49]. Let \mathcal{I} be a finite set of processes, also called instances. For any instance $i \in \mathcal{I}$, the set Σ_i^{int} denotes a finite set of *internal actions*, $\Sigma_i^! = \{i!j | j \in \mathcal{I} \setminus \{i\}\}$ a set of *send actions* and $\Sigma_i^? = \{i?j | j \in \mathcal{I} \setminus \{i\}\}$ a set of *receive actions*. The alphabet of events associated with the instance i is the disjoint union of these three sets: $\Sigma_i = \Sigma_i^{int} \cup \Sigma_i^! \cup \Sigma_i^?$. We shall assume that the alphabets Σ_i are disjoint and let $\Sigma_{\mathcal{I}} = \bigcup_{i \in \mathcal{I}} \Sigma_i$. Given an action $a \in \Sigma_{\mathcal{I}}$, $Ins(a)$ denotes the unique instance i such that $a \in \Sigma_i$. Finally, for any partial order $po = (V, E, l)$ whose vertices are labeled over $\Sigma_{\mathcal{I}}$, we denote by $Ins(v)$ the instance on which the event $v \in V$ occurs: $Ins(v) = Ins(l(v))$.

Definition 8 (Message Sequence Chart (MSC)). *A message sequence chart is a partial order $M = (V, \leq, l)$ over $\Sigma_{\mathcal{I}}$ such that*

- *Events occurring on the same process are linearly ordered: For every pair of events $v, v' \in V$ if $Ins(v) = Ins(v')$ then $v \leq v'$ or $v' \leq v$.*
- *For any two distinct processes i, j , there are as many send events from i to j as receive events of j from i : $\#^{i!j}(V) = \#^{j?i}(V)$.*
- *The n -th message sent from i to j is received when the n -th event $j?i$ occurs, i.e., the channels are assumed to be FIFO. $l(v) = i!j$ and $l(v') = j?i$ and $\#^{i!j}(\downarrow v) = \#^{j?i}(\downarrow v')$ then $v \leq v'$.*

- If $v \prec v'$ and $Ins(v) \neq Ins(v')$ then $l(v) = i!j$, $l(v') = j?i$ and $\#^{i!j}(\downarrow v)$ is equal to $\#^{j?i}(\downarrow v')$.

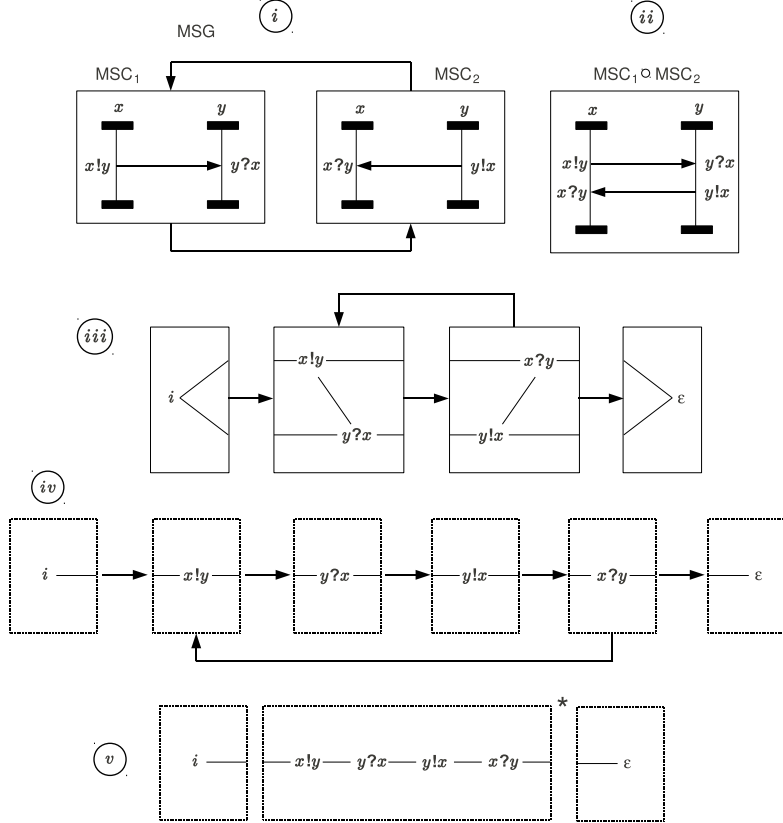


Figure 6: i) MSG whose vertices are labeled with MSC_1 and MSC_2 . ii) Composition of $MSC_1 \circ MSC_2$. III) Direct translation from MSG to a slice graph which is not transitive reduced. iv) Canonical slice graph generating the same set of partial orders. v) An elegant way of specifying the same language using a slice expression.

The composition $M \circ M'$ of two MSCs $M = (V, \leq, l)$ and $M' = (V', \leq', l')$ can be defined directly into the partial order level as the transitive closure of the graph

$$M \cup M' \cup \{(v, v') \in V \times V' | Ins(v) = Ins(v')\}.$$

The partial order language generated by a message sequence graph \mathcal{M} is the set $\mathcal{L}_{PO}(\mathcal{M})$ of all partial orders obtained by the composition of sequences of MSCs which labels walks in \mathcal{M} . A language \mathcal{L} of MSCs is linearization-regular [33] if its set of linearizations $lin(\mathcal{L}) = \cup_{M \in \mathcal{L}} lin(M)$ is recognizable in the free monoid Σ_T^* . The connectivity graph of a MSC M is the graph $C(M)$ whose vertices are the instances of M and there is an edge from instance i to instance j if i sends some message to j . An MSG $\mathcal{M} = (\mathcal{V}, \mathcal{E}, l)$

is locally synchronized [50] (called bounded in [2]) if for each loop $w = v_1 v_2 \dots v_n v_1$ in \mathcal{M} the connectivity graph of the MSC $\mathsf{l}(v_1) \circ \mathsf{l}(v_2) \circ \dots \circ \mathsf{l}(v_n) \circ \mathsf{l}(v_1)$ has a unique strongly connected component. It can be proved that an MSC language generated by a MSG is linearization-regular if and only if it is locally synchronized [33]. In the next lemma we prove that the partial order language of any MSG can be represented by a slice graph. Furthermore, locally synchronized MSGs correspond to loop-connected slice graphs, which can be saturated by Theorem 6.

Lemma 8 (From MSCs to Slices). *Let \mathcal{M} be a message sequence graph. Then there exists a slice graph $\mathcal{SG}_{\mathcal{M}}$ satisfying $\mathcal{L}_{PO}(\mathcal{M}) = \mathcal{L}_{PO}(\mathcal{SG}_{\mathcal{M}})$. Furthermore if $\mathcal{L}_{PO}(\mathcal{M})$ is linearization-regular then $\mathcal{SG}_{\mathcal{M}}$ is loop-connected.*

Proof. We associate to each MSC M a slice \mathbf{S}_M in such a way that for each two MSCs M_1 and M_2 , the partial order $M_1 \cdot M_2$ is equal to the partial order induced by the transitive closure of $\mathbf{S}_{M_1} \circ \mathbf{S}_{M_2}$ (modulo the frontier vertices). Each frontier of \mathbf{S}_M will have $|\mathcal{I}|$ nodes, one for each instance $i \in \mathcal{I}$. If M is a MSC then the slice \mathbf{S}_M is the Hasse diagram of M together with the new frontier vertices and some new edges which we describe as follows: If for some instance $i \in \mathcal{I}$, there is no vertex v of M such that $\text{Ins}(v) = i$, then we add an edge from the i -th in-frontier of \mathbf{S}_M to the i -th out-frontier of M . For all the other instances in \mathcal{I} , add an edge from the i -th in-frontier vertex of \mathbf{S}_M to the unique minimal vertex v of M satisfying $\text{Ins}(v) = i$, and an edge from the unique maximal vertex v' of M satisfying $\text{Ins}(v') = i$ to the i -th out-frontier vertex of \mathbf{S}_M . We observe that although each slice \mathbf{S}_M is transitive reduced, the composition $\mathbf{S}_{M_1} \circ \mathbf{S}_{M_2}$ is not necessarily transitive reduced (Fig. 6.iii). Now if the partial order language represented by \mathcal{M} is linearization-regular, then \mathcal{M} is locally synchronized. Furthermore each sequence $M_1 M_2 \dots M_n M_1$ of MSCs labeling a loop in \mathcal{M} corresponds to a sequence $\mathbf{S}_{M_1} \mathbf{S}_{M_2} \dots \mathbf{S}_{M_n} \mathbf{S}_{M_1}$ labeling a loop in \mathcal{SG} . One can then verify that if the connectivity graph of $M_1 \circ M_2 \circ \dots \circ M_n$ has a unique strongly connected component, then gluing the out frontier of the slice $\mathbf{S}_{M_1} \circ \mathbf{S}_{M_2} \circ \dots \circ \mathbf{S}_{M_n}$ with its own in frontier we also have a unique strongly connected component, and thus \mathcal{SG} is loop-connected. \square

6.2.1. Comparison between MSC languages and Slice languages

A special property which is satisfied by MSC's, and which is also observed in partial orders represented by Mazurkiewicz trances is the following: If M, M' are two partial orders represented through MSC's (or through Mazurkiewicz traces) then M is isomorphic to M' if and only if $\text{lin}(M) \cap \text{lin}(M') = \emptyset$ [49, 33] where $\text{lin}(M)$ denotes the set of linearizations of M . This property which is fundamental for the development of several aspects of MSC-language theory and Mazurkiewicz trace theory, turns also to be a bottleneck for their expressiveness. For instance, some very simple partial order languages, such as the one depicted in Figure 7.i cannot be represented by any formalisms satisfying this property. This bottleneck is not a issue when dealing with slices languages because the role of linearization of a partial order is completely replaced by the notion of unit decomposition of their Hasse diagrams.

A notion of atomic MSC has also been defined: An MSC M is a component of a MSC M' if there exist MSCs M_1 and M_2 such that $M' = M_1 \circ M \circ M_2$. M is an atomic MSC if the only component of M is M itself. Two atomic MSCs M_1 and M_2 if their vertices are labeled with actions from disjoint sets of processes (instances). We notice however

The figure contains two Feynman diagrams. The left diagram shows a fermion line (represented by a solid line with arrows) entering from the left, passing through a self-energy loop. The loop consists of a fermion line and a gluon line (represented by a dashed line). The right diagram shows a similar setup, but the self-energy loop is more complex, involving multiple gluon lines and a ghost line (represented by a dotted line).

that not every MSC can be decomposed into atomic MSC's consisting on a unique event, or more appropriately, consisting on a unique message being sent and received. Contrast this with the fact that any DAG can be written as a composition of unit slices.

7. Applications to Petri Nets

7.1. Partial order Semantics of p/t-nets

27

as $m'(p) = m(p) - \hat{p}(t) + \check{p}(t)$. The initial marking m_0 of N is given by $m_0(p) = p_0$ for each $p \in P$. A sequence of transitions $t_0 t_1 \dots t_{n-1}$ is an occurrence sequence of N if there exists a sequence of markings $m_0 m_1 \dots m_n$ such that t_i is enabled at m_i and if m_{i+1} is obtained by the firing of t_i at marking m_i . A marking m is legal if it is the result of the firing of an occurrence sequence of N . A place p of N is k -safe if $m(p) \leq k$ for each legal marking m of N . A net N is k -safe if each of its places is k -safe. N is bounded if it is k -safe for some k . The union of two p/t -nets $N_1 = (P_1, T)$ and $N_2 = (P_2, T)$ having a common set of transitions T is the p/t -net $N_1 \cup N_2 = (P_1 \dot{\cup} P_2, T)$. We consider that the multiplicity of a place p in $P_1 \dot{\cup} P_2$ is the sum of its multiplicities in P_1 and in P_2 .

Definition 9 (Process). A process of a p/t -net $N = (P, T)$ is a DAG $\pi = (B \dot{\cup} V, F, \rho)$ where the vertex set $B \dot{\cup} V$ is partitioned into a set of conditions B and a set of events V . $F \subseteq (B \times V) \cup (V \times B)$ and $\rho : (B \cup V) \rightarrow (P \cup T) \cup \{\iota, \epsilon\}$ are required to satisfy the following conditions:

1. π has a unique minimal vertex $v_\iota \in V$ and a unique maximal vertex $v_\epsilon \in V$.
2. Conditions are unbranched: $\forall b \in B, |\{(b, v) \in F\}| = 1 = |\{(v, b) \in F\}|$.
3. Places label conditions and transitions label events. Minimal and maximal vertices have special labels.

$$\rho(B) \subseteq P \quad \rho(V \setminus \{v_\iota, v_\epsilon\}) \subseteq T \quad \rho(v_\iota) = \iota \quad \rho(v_\epsilon) = \epsilon$$

4. If ρ labels an event $v \in V \setminus \{v_\iota, v_\epsilon\}$ with a transition $t \in T$ then for each $p \in P$, v has $\hat{p}(t)$ preconditions and $\check{p}(t)$ postconditions labeled by p :

$$|\{(b, v) \in F : \rho(b) = p\}| = \hat{p}(t) \quad |\{(v, b) \in F : \rho(b) = p\}| = \check{p}(t)$$

5. For each $p \in P$, v_ι has p_0 post-conditions labeled by p : $|\{(v_\iota, b) : \rho(b) = p\}| = p_0$.

The only point our definition of process differs from the usual definition of p/t -net process [30] is the addition of a minimal event v_ι which is labeled with a letter $\iota \notin T$ and a maximal event v_ϵ which is labeled with a letter $\epsilon \notin T$. We notice that item 9.2 implies that every condition which is not connected to an event $v \in V$ labeled by a transition $t \in T$, is necessarily connected to v_ϵ . Intuitively, ι loads the initial marking of N and ϵ empties the marking of N after the occurrence of all events of the process. We call attention to the fact that the number of conditions connected to v_ϵ varies according to the process.

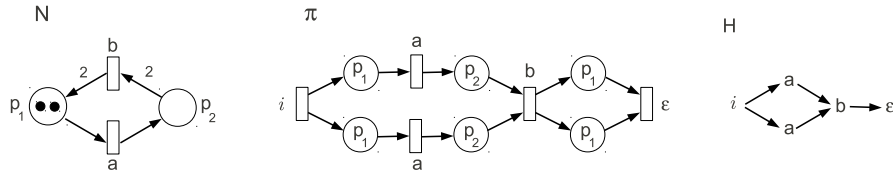


Figure 8: A p/t -net N , a process π of N and the Hasse diagram of the causal order induced by π .

A sequentialization of a partial order $po = (V, <, l)$ is another partial order $po' = (V, <', l)$ satisfying $< \subseteq <'$. The causal order of a process π is obtained from it by abstracting

its conditions and by considering the partial order induced by its events. An execution is a sequentialization of a causal order.

Definition 10 (Causal Orders and Executions of p/t -net Processes). *The causal order of a process $\pi = (B \dot{\cup} V, F, \rho)$ of a p/t -net N is the partial order $po_\pi = (V, <, l)$ where $< = F^*|_{V \times V}$ and $l = \rho|_V$. An execution of π is a sequentialization of po_π .*

We denote $\mathcal{L}_{cau}(N)$ the set of all causal orders derived from processes of N , $\mathcal{L}_{ex}(N)$ the set of all its executions, and write simply $\mathcal{L}_{PO}(N)$ whenever it is not relevant whether we are representing the set of causal orders or the set of executions of N .

7.2. Interlaced Flows, Executions and Causal Orders

Let $N = (P, T)$ be a p/t -net, $H = (V, E, l)$ a Hasse diagram with $l : V \rightarrow T$, and $p \in P$ be a place of N . Then a p -interlaced flow on H with respect to N is a four tuple $f = (\mathbf{bb}, \mathbf{bf}, \mathbf{pb}, \mathbf{pf})$ of functions of type $E \rightarrow \mathbb{N}$ whose components satisfy the three following equations around each vertex v of H :

$$\sum_{e^t=v} \mathbf{bf}(e) + \mathbf{pf}(e) = \sum_{e^s=v} \mathbf{pb}(e) + \mathbf{pf}(e) \quad (6)$$

$$In(v) = \sum_{e^t=v} \mathbf{bb}(e) + \mathbf{pb}(e) = \hat{p}(l(v)) \quad (7)$$

$$Out(v) = \sum_{e^s=v} \mathbf{bb}(e) + \mathbf{bf}(e) = \check{p}(l(v)) \quad (8)$$

Intuitively, for each $e \in E$, $\mathbf{pb}(e)$ counts some of the tokens produced in the **past** of e^s and consumed **by** e^t ; $\mathbf{pf}(e)$, some of the tokens produced in the **past** of e^s and consumed in the **future** of e^t , and $\mathbf{bf}(e)$, some of the tokens produced **by** e^s and consumed in the future of e^t . Thus equation 6 states that on interlaced flows, the total number of tokens produced in the past of a vertex v , that arrives at it without being consumed, will eventually be consumed in the future of v . The component $\mathbf{bb}(e)$, counts the total number of tokens produced **by** e^s and consumed **by** e^t . Thus, equation 7 states that the total number of tokens consumed by v is equal to $\hat{p}(l(v))$ while equation 8 states that the total number of tokens produced by v is $\check{p}(l(v))$. Interlaced flows were introduced in [16] to characterize Hasse diagrams of executions and causal orders of p/t -nets. This characterization is formalized below in Theorem 7. Intuitively it says that a Hasse diagram H induces an execution of a given p/t -net N , if and only if it can be associated to a set of p -interlaced flows, one for each place p of N . A similar result holds with respect to Hasse diagrams of causal orders of N . The only difference is that if an edge e belongs to the Hasse diagram of a causal order of N , then it must have arisen from a token that was transmitted from the event that labels its source vertex to the event that labels its target vertex, by using some place $p \in P$ as a channel. Thus in the flow that corresponds to p , the component which is responsible for the direct transmission of tokens must be strictly greater than zero.

Theorem 7 (Interlaced Flow Theorem[16]). *Let $N = (P, T)$ be a (not necessarily bounded) p/t -net and $H = (V, E, l)$ be a Hasse diagram. Then*

- (i) *The partial order induced by H is an execution of N iff there exists a p -interlaced flow $f_p : E \rightarrow \mathbb{N}^4$ in H for each place p .*
- (ii) *The partial order induced by H is a causal order of N iff there exists a set $\{f_p\}_{p \in P}$ of p -interlaced flows such that for every edge e of H , the component $\mathbf{bb}_p(e)$ of $f_p(e)$, which denotes the direct transmission of tokens, is strictly greater than zero for at least one $p \in P$.*

By using Theorem 7 we are able to provide a sliced characterization of executions and causal orders of p/t -nets. Namely, let $N = (P, T)$ be a p/t -net, $p \in P$ be a place of N and $\mathbf{S}_1 \mathbf{S}_2 \dots \mathbf{S}_n$ be a unit decomposition of a Hasse diagram H such that $\mathbf{S}_i = (\{v_i\}, E_i, l_i)$. Then a p -flow coloring of $\mathbf{S}_1 \mathbf{S}_2 \dots \mathbf{S}_n$ is a sequence of functions $f_p^1 f_p^2 \dots f_p^n$ with $f_p^i : E_i \rightarrow \mathbb{N}$ such that for any two consecutive slices $\mathbf{S}_i \mathbf{S}_{i+1}$, it holds that the value associated by f_p^i to each edge e touching the out frontier of \mathbf{S}_i is equal to the value associated by f_p^{i+1} to its corresponding edge touching the in-frontier of \mathbf{S}_{i+1} . We notice that a Hasse diagram H has a p -interlaced flow f_p with respect to N if and only if each unit decomposition $\mathbf{S}_1 \mathbf{S}_2 \dots \mathbf{S}_n$ of $H = (V, E, l)$ admits a p -flow coloring $f_p^1 f_p^2 \dots f_p^n$. To see this, for each $e \in \mathbf{S}_i$ that is the sliced part of an edge $e' \in H$, set $f_p(e) = f_p(e')$. In this way it makes sense to say that each f_p^i is a *sliced p -interlaced flow* for \mathbf{S}_i .

Now an *execution coloring* of a unit decomposition $\mathbf{S}_1 \mathbf{S}_2 \dots \mathbf{S}_n$ of a Hasse diagram with respect to a p/t -net N is a sequence $F_1 F_2 \dots F_n$ where each $F_i = \{f_p^i\}_{p \in P}$ is a set of sliced p -interlaced flows for \mathbf{S}_i where for each i with $1 \leq i \leq n-1$ and each $p \in P$, the value associated by f_p^i to each edge e touching the out-frontier of \mathbf{S}_i is equal to the value associated by f_p^{i+1} to its corresponding edge e' touching the in-frontier of \mathbf{S}_{i+1} . A *causal coloring* of $\mathbf{S}_1 \mathbf{S}_2 \dots \mathbf{S}_n$, is an execution coloring $F_1 F_2 \dots F_n$ with the additional requirement that for each i , and each edge e in \mathbf{S}_i , there is a $p \in P$ such that the component \mathbf{bb}_p^i of $f_p^i \in F_i$ accounting for the direct transmission of tokens is strictly greater than 0. Using the same argument as above we have that $\mathbf{S}_1 \mathbf{S}_2 \dots \mathbf{S}_n$ is a unit decomposition of an execution (causal order) H of N if and only if it admits an execution (causal) coloring $F_1 F_2 \dots F_n$. We call each F_i , a *sliced execution-flow* (*sliced causal-flow*) for \mathbf{S}_i . The following proposition will be important for our refined characterization.

Proposition 2. *Let $N = (P, T)$ be a k -bounded p/t -net, H be the Hasse diagram of an execution of N , $\mathbf{S}_1 \mathbf{S}_2 \dots \mathbf{S}_n$ be a unit decomposition of H where $\mathbf{S}_i = (\{v_i\}, E_i, l_i)$, and $m_i : P \rightarrow \mathbb{N}$ be the marking of N after the firing of the transitions $l(v_1)l(v_2) \dots l(v_i)$. Then*

- (i) *if $F_1 F_2 \dots F_n$ is an execution coloring of $\mathbf{S}_1 \mathbf{S}_2 \dots \mathbf{S}_n$ then for each $f_p^i : E_i \rightarrow \mathbb{N}^4 \in F_i$ with $f_p^i(e) = (\mathbf{bb}_p^i(e), \mathbf{bf}_p^i(e), \mathbf{pb}_p^i(e), \mathbf{pf}_p^i(e))$ for $e \in E_i$, the following equation is satisfied*

$$\sum_{e'} \mathbf{bb}_p^i(e') + \mathbf{bf}_p^i(e') + \mathbf{pb}_p^i(e') + \mathbf{pf}_p^i(e') = m_i(p) \quad (9)$$

where the sum is over all edges e' touching the out-frontier of \mathbf{S}_i .

- (ii) *if $F_1 F_2 \dots F_n$ is a causal coloring of F then for each i , the size of the out-frontier of \mathbf{S}_i is at most $k|P|$.*

Intuitively, Proposition 2.i says that in a sliced execution flow, for each place $p \in P$ the sum of all tokens attached to the edges of the out-frontier of each unit slice \mathbf{S}_i is equal to the number of tokens at place p after the execution of the firing sequence $l(v_1)l(v_2)...l(v_i)$, where for $1 \leq j \leq i$, v_j is the center vertex of \mathbf{S}_j . Also notice that in a k -bounded p/t -net with $|P|$ places at most $k|P|$ tokens may be present in the whole net after each firing sequence. Thus, Proposition 2.ii follows from Proposition 2.i together with the fact that in a causal coloring $F_1F_2...F_n$ for each i and edge $e \in \mathbf{S}_i$, the component $\mathbf{bb}_p^i(e)$ must be strictly greater than 0 for at least one $p \in P$. \square

Theorem 8 (Refined Expressibility Theorem). *Let N be a k -bounded p/t -net. Then*

- (i) *For any $c \geq 1$ there exists a (not necessarily saturated) Hasse diagram generator $\mathcal{HG}_{ex}^{c\exists}$ over Σ_S^c representing all executions of N of existential slice width at most c .*
- (ii) *For any $c \geq 1$ there exists a canonical saturated Hasse diagram generator \mathcal{HG}_{ex}^c over Σ_S^c representing all executions of N of global slice width at most c .*
- (iii) *for any $c \geq 1$ there exists a canonical saturated Hasse diagram generator \mathcal{HG}_{cau}^c over Σ_S^c representing all the causal orders of N of global slice width at most c . Furthermore, for any $c \geq k|P|$, we have that $\mathcal{L}_{PO}(\mathcal{HG}_{cau}^c) = \mathcal{L}_{PO}(\mathcal{HG}_{cau}^{k|P|})$.*

Proof. (i) Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{S})$ be a slice graph over Σ_S^c where for each unit slice $\mathbf{S} \in \Sigma_S^c$, we have a vertex $\mathbf{v}_{\mathbf{S}} \in \mathcal{V}$ with $\mathcal{S}(\mathbf{v}_{\mathbf{S}}) = \mathbf{S}$, and where there is an edge $(\mathbf{v}_{\mathbf{S}}, \mathbf{v}_{\mathbf{S}'})$ if and only if \mathbf{S} can be glued to \mathbf{S}' . Then clearly, a graph G is in $\mathcal{L}_G(\mathcal{G})$ if and only if it has existential slice width at most c . Also let $\mathcal{HG} = (\mathcal{V}', \mathcal{E}', \mathcal{S}')$ be the transitive reduced version of \mathcal{G} . At this point we have that a partial order po is in $\mathcal{L}_{PO}(\mathcal{HG})$ if and only if its Hasse diagram has existential slice width c . Now we will describe how to filter out from $\mathcal{L}_{PO}(\mathcal{HG})$ all the partial orders that are not executions of N : Define the Hasse diagram generator $filter_{ex,c}^N(\mathcal{HG}) = (\mathcal{V}'', \mathcal{E}'', \mathcal{S}'')$ as follows: For each vertex $\mathbf{v}_{\mathbf{S}} \in \mathcal{V}$ and each sliced execution flow $F = \{f_p\}_{p \in P}$ of \mathbf{S} such that for each $p \in P$, $\sum_e f_p(e) \leq c$ where e ranges over the edges touching the out-frontier of \mathbf{S} , we add a vertex $\mathbf{v}_{\mathbf{S},F}$ to \mathcal{V}'' and label this vertex with \mathbf{S} . Furthermore, we add an edge $(\mathbf{v}_{\mathbf{S},F}, \mathbf{v}_{\mathbf{S}',F'})$ to \mathcal{E}'' if and only if $(\mathbf{v}_{\mathbf{S}}, \mathbf{v}_{\mathbf{S}'}) \in \mathcal{E}$ and if (\mathbf{v}, F) can be glued to (\mathbf{v}', F') . In this way, there is a path $\mathbf{v}_{\mathbf{S}_1, F_1} \mathbf{v}_{\mathbf{S}_2, F_2} ... \mathbf{v}_{\mathbf{S}_n, F_n}$ from an initial to a final vertex in $filter_{ex,c}^N(\mathcal{HG})$ if and only if $F_1F_2...F_n$ is an execution coloring of $\mathbf{S}_1\mathbf{S}_2...\mathbf{S}_n$. By Theorem 7.i such a coloring exists if and only if the Hasse diagram $H = \mathbf{S}_1 \circ \mathbf{S}_2 \circ ... \circ \mathbf{S}_n$ has existential slice width at most c and if its induced partial order is an execution of N . Therefore we can set $\mathcal{HG}_{ex}^{c\exists}(N)$ to be $filter_{ex,c}^N(\mathcal{HG})$.

(ii) Let \mathcal{HG}^c be the Hasse diagram generator of Lemma 5 whose graph language consists precisely of the Hasse diagrams of global slice width at most c . Now let \mathcal{HG}^\cap be the Hasse diagram generator whose graph language consists in the intersection $\mathcal{L}_G(filter_{ex,c}^N(\mathcal{HG})) \cap \mathcal{L}_G(\mathcal{HG}^c)$. By Lemma 6.1, \mathcal{HG}^\cap can be effectively constructed from \mathcal{HG}^c and from $filter_{ex,c}^N(\mathcal{HG})$, observe that since $filter_{ex,c}^N(\mathcal{HG})$ is not saturated, it is not evident that \mathcal{HG}^\cap is saturated. Nevertheless, the saturability of \mathcal{HG}^\cap follows from Theorem 7.i and from the fact that whenever a set $F = \{f_p\}_{p \in P}$ of p -interlaced flows can be associated to a Hasse diagram H , we have that any unit decomposition $\mathbf{S}_1\mathbf{S}_2...\mathbf{S}_n$ of H admit an execution coloring $F_1F_2...F_n$, where each F_i is the sliced part of F that is associated to \mathbf{S}_i . Since \mathcal{HG}^\cap is saturated, by Theorem 4.3 there is a canonical Hasse diagram generator $\mathcal{HG}_{ex}^c = \mathcal{C}(\mathcal{HG}^\cap)$ representing the same graph language \mathcal{HG}^\cap , and consequently the same partial order language.

(iii) The proof is analogous to the proof of item (ii), except for two small adaptations: Replace each occurrence of the word "execution" by the word "causal", and each occurrence of the subscript ex by the subscript cau . In this way whenever $F_1 F_2 \dots F_n$ appears in the proof it will denote a causal coloring instead of an execution coloring. Analogously the filter $filter_{cau,c}^N(\mathcal{HG})$ will filter out from the graph language of \mathcal{HG} all the Hasse diagrams whose induced partial order is not a causal order of N . The only additional caveat, is that since the Hasse diagram of any causal order of N has global slice width at most $k|P|$, we have that $\mathcal{L}_{PO}(\mathcal{HG}_{cau}^c) = \mathcal{L}_{PO}(\mathcal{HG}_{cau}^{k|P|})$ whenever $c \geq k|P|$. Therefore the whole causal order behavior of N can be represented by $\mathcal{HG}_{cau}^{k|P|}$. \square

We observe it may be the case that the graph language of $\mathcal{L}_G(\mathcal{HG}_{ex}^c(N)) \subseteq \mathcal{L}_G(\mathcal{HG}_{ex}^{c+1}(N))$ for every $c + 1$. And indeed this fact can already be noticed in the execution behavior of rather simple nets such as the one depicted in Figure 9.i. While the causal behavior $\mathcal{L}_{cau}(N)$ of N , which is intuitively depicted in Figure 9, is relatively simple, and can be easily described in terms of regular slice languages (Figure 9.iii), the set $\mathcal{L}_{ex}(N)$ of all sequentializations of partial orders in $\mathcal{L}_{cau}(N)$, contains subfamilies of partial orders whose Hasse diagrams have unbounded existential slice width, and that for this reason, cannot be represented through slice languages over finite alphabets. An example of such a subfamily is the set of partial orders induced by the sequence $\{H_n = (V_n, E_n, l_n)\}_{n \in \mathbb{N}}$ of Hasse diagrams defined below and depicted in Figure 9.iv.

$$\begin{aligned} V_n &= \{v_l, v_\varepsilon\} \cup \{v_{a_1}, \dots, v_{a_n}\} \cup \{v_{b_1}, \dots, v_{b_n}\} \\ l(v_l) &= \iota, l(v_\varepsilon) = \varepsilon, l(v_{a_i}) = a, l(v_{b_i}) = b \\ E_n &= \{(v_l, v_{a_1}), (v_l, v_{b_1}), (v_{a_n}, v_\varepsilon), (v_{b_n}, v_\varepsilon)\} \cup \\ &\quad \{(v_{a_i}, v_{a_{i+1}})\} \cup \{(v_{b_i}, v_{b_{i+1}})\} \cup \\ &\quad \{(v_{a_i}, v_{b_{n-i}})\} \cup \{(v_{b_i}, v_{a_{n-i}})\} \end{aligned}$$

Therefore, the parametrization of the language of executions of a p/t -net with respect to the maximal global slice width of the respective Hasse diagrams is unavoidable, if we are willing to represent executions via regular slice languages. When dealing with the causal behavior of bounded p/t -nets such a parametrization is not essential since the whole causal behavior of k -bounded p/t -nets with set of places P can already be captured by regular slice languages over $\Sigma_{\mathbb{S}}^{k|P|}$. In particular, a neat implication of Theorem 8.iii is that if a k -bounded p/t -net $N = (P, T)$ and a k' -bounded p/t -net $N' = (P', T)$ have the same partial order behavior, then $\mathcal{HG}_{cau}^{k|P|}(N) \simeq \mathcal{HG}_{cau}^{k'|P'|}(N')$.

Theorem 9 (p/t -nets and Hasse diagram generators [16]).

Let \mathcal{L}_{PO} be a partial order language generated by a (not necessarily saturated) Hasse diagram generator \mathcal{HG} over $\Sigma_{\mathbb{S}}^c$.

- **Verification:** Let N be a bounded p/t -net. Then the following problems are decidable:

- Is $\mathcal{L}_{ex}(N) \cap \mathcal{L}_{PO} = \emptyset$? (Is $\mathcal{L}_{cau}(N) \cap \mathcal{L}_{PO} = \emptyset$?)
- Is $\mathcal{L}_{PO} \subseteq \mathcal{L}_{ex}(N)$? (Is $\mathcal{L}_{PO} \subseteq \mathcal{L}_{cau}(N)$?)

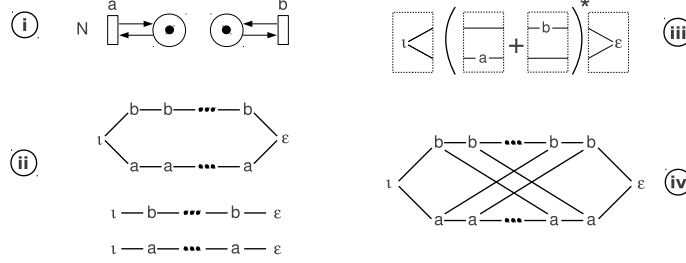


Figure 9: i) A 1-bounded p/t -net N ii) The causal language $\mathcal{L}_{cau}(N)$ of N iii) A weakly saturated (but not yet transitive reduced) slice language representing $\mathcal{L}_{cau}(N)$ iv) An intuitive depiction of a family of partial orders that is included in the set $\mathcal{L}_{ex}(N)$ of sequentializations of partial orders in $\mathcal{L}_{cau}(N)$, but that is not representable through slice languages (not even non-saturated slice languages) over a finite slice alphabet.

- **Synthesis:** Let $k \geq 1$ and $r \geq 1$. Then it is possible to determine if it exists, and if so, automatically synthesize
 - a k -bounded p/t -net N whose execution behavior $\mathcal{L}_{ex}(N)$ minimally includes \mathcal{L}_{PO} .
 - a k -bounded p/t -net N with place repetition number r whose causal behavior $\mathcal{L}_{cau}(N)$ minimally includes \mathcal{L}_{PO} .

We point out that in order to carry the verification result stated in Theorem 9 we do not need to construct the Hasse diagram generators $\mathcal{HG}_{ex}^{c\exists}, \mathcal{HG}_{ex}^c$ and \mathcal{HG}_{cau}^c . Instead we may apply the filters used in the proof of Theorem 8 directly to the Hasse diagram generator we want to verify. In what follows let $sem = ex$ if we are performing the verification according to the execution semantics and $sem = cau$ if we are performing the verification according to the causal semantics. Then we have that $\mathcal{L}_{sem}(N) \cap \mathcal{L}_{PO}(\mathcal{HG}) = \emptyset$ if and only if the slice language of the filtered version of \mathcal{HG} is empty, i.e., if $\mathcal{L}(\text{filter}_{sem,c}^N(\mathcal{HG})) = \emptyset$, while $\mathcal{L}_{PO}(\mathcal{HG}) \subseteq \mathcal{L}_{ex}(N)$ if and only if the slice language of the filtered version of \mathcal{HG} equals the slice language of \mathcal{HG} itself, i.e., if $\mathcal{L}(\text{filter}_{sem,c}^N(\mathcal{HG})) = \mathcal{L}(\mathcal{HG})$. Since the slice languages represented by Hasse diagram generators are regular, and since emptiness and equality are decidable for regular languages, the verification results hold.

Concerning the synthesis with respect to the causal semantics we need to specify à priori, the maximum number r of repeated copies a place is allowed to have in the synthesized net. This additional parameter is not necessary when considering the synthesis with respect to the execution semantics because the set of executions of a p/t -net remains invariant upon the addition of a place that is already part from the net. Adding a repeated place to a p/t -net may however increase its causal behavior by increasing the possibility of causal interactions between its transitions. We refer to [16] for a detailed discussion on this topic.

We finish this section by stating a corollary (Corollary 3) that extends the applicability of slice graphs and Hasse diagram generators by showing that they can serve as an interface between p/t -nets and other well known formalisms for the specification of partial order languages, such as MSC -languages and Mazurkiewicz trace languages. More precisely, it

states that both the *verification* and the *synthesis* results of Theorem 9 can be reformulated in terms of these formalisms. Indeed, as we showed in Section 6 both formalisms can be mapped to slice graphs, which in general generate non transitive reduced slice languages. By an application of our transitive reduction algorithm (Theorem 2) these slice graphs can be transformed into Hasse diagram generators, with the aim to meet the requirements of Theorem 9 (which is not valid for general slice graphs). Finally these Hasse diagram generators can be used to address both the verification and the synthesis of p/t -nets as stated in Theorem 9.

Corollary 3 (MSC Languages, Mazurkiewicz Traces and p/t -nets). *The synthesis and verification results stated in Theorem 9 is equally valid if the partial order language \mathcal{L}_{PO} is represented by a pair (\mathcal{A}, I) of finite automaton and independence relation, or by a message sequence graph \mathcal{M} .*

We emphasize that the verification and synthesis results of Theorem 9 do not require the Hasse diagrams to be saturated. Analogously, Corollary 3 does not require the language specified by the pair (\mathcal{A}, I) to be recognizable (i.e. the trace closure of $\mathcal{L}(\mathcal{A})$ to be recognized by a finite automaton), nor the partial order language specified by the message sequence graph \mathcal{M} to be linearization-regular. For a matter of comparison we point out that the synthesis of labeled p/t -nets (i.e., nets in which two transitions may be labeled by the same action) from recognizable Mazurkiewicz trace languages (and indeed more generally from the local trace languages [35]) was addressed in [36, 42]. Corollary 3 concerns the synthesis of unlabeled p/t -nets. To contrast the partial order behavior of labeled and unlabeled p/t -nets, we notice that labeled 1-safe p/t -nets are already as partial order expressive as their b -bounded counterparts [8], while this is not the case for unlabeled p/t -nets. Thus the synthesis of unlabeled nets which is addressed in Corollary 3 tends to be harder.

8. Final Comments

The main contributions of the present work were twofold. First, we devised an algorithm that transitive reduces any slice graph into a Hasse diagram generator representing the same set of partial orders. Second, we developed the theory of saturated slice languages, which lifts some of the most intuitive aspects of trace theory to the slice setting. From a conceptual perspective our transitive reduction algorithm conciliates the flexibility of reasoning about partial orders in terms of DAGs, which is implicit in most of the literature dedicated to the representation of infinite families of partial orders, with the aesthetical and algorithmic advantages of specifying such families through sets of Hasse diagrams. From a practical perspective our algorithm turned to be a necessary step towards putting distinct concurrency theoretic formalisms such as Mazurkiewicz traces, message sequence charts and Petri nets, into a common ground with respect to the partial order languages they represent. As a consequence we were able to address the verification and automatic synthesis of concurrent systems from an unified perspective. By combining our transitive reduction with our development of saturated slice languages we were able to address the canonization of slice graphs with respect to their partial order languages and to prove several decidability and computability results respective to the manipulation of

these slice graphs. Furthermore we showed via reductions that all these results hold as well for partial order languages represented by recognizable Mazurkiewicz trace languages, by recognizable MSC languages specified by Message sequence graphs, and by Petri nets. Therefore we consider that the overall contribution of the present work consists in a robust methodology to compare and operate with partial order languages generated by seemingly disconnected formalisms.

References

- [1] I. J. Aalbersberg and H. J. Hoogeboom. Characterizations of the decidability of some problems for regular trace languages. *MST: Mathematical Systems Theory*, 22, 1989.
- [2] R. Alur and M. Yannakakis. Model checking of message sequence charts. In *Proc. of the 10th International Conference on Concurrency Theory (CONCUR)*, volume 1664 of *LNCS*, pages 114–129. LNCS, Springer-Verlag, 1999.
- [3] E. Badouel and P. Darondeau. On the synthesis of general Petri nets. Technical Report PI-1061, IRISA, 1996.
- [4] E. Badouel and P. Darondeau. Theory of regions. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *LNCS*, pages 529–586, 1998.
- [5] M. Bauderon and B. Courcelle. Graph expressions and graph rewritings. *Mathematical Systems Theory*, 20(2-3):83–127, 1987.
- [6] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Synthesis of Petri nets from finite partial languages. *Fundamenta Informaticae*, 88(4):437–468, 2008.
- [7] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Synthesis of Petri nets from infinite partial languages. In *Proc. of the 8th International Conference in Application of Concurrency to System Design*, pages 170–179. IEEE, 2008.
- [8] E. Best and H. Wimmel. Reducing k -safe Petri nets to pomset-equivalent 1-safe Petri nets. In *Proc. of 21th International Conference in Applications and Theory of Petri Nets (ICATPN)*, volume 1825 of *LNCS*, pages 63–82, 2000.
- [9] F. Bossut, M. Dauchet, and B. Warin. A kleene theorem for a class of planar acyclic graphs. *Information and Computation*, 117(2):251–265, 1995.
- [10] Bozapalidis and Kalampakas. Recognizability of graph and pattern languages. *Acta Informatica*, 42, 2006.
- [11] F.-J. Brandenburg and K. Skodinis. Finite graph automata for linear and boundary graph languages. *Theoretical Computer Science*, 332(1-3):199–232, 2005.
- [12] A. Church. Logic, arithmetic and automata. In *Proc. of the International Congress of Mathematicians*, pages 23–35, 1962.
- [13] B. Courcelle. Graph expressions and graph rewritings. *Mathematical Systems Theory*, 20:83–127, 1987.

- [14] P. Darondeau. Deriving unbounded Petri nets from formal languages. *LNCS*, 1466:533–548, 1998.
- [15] P. Darondeau. Region based synthesis of P/T-nets and its potential applications. In *Proc. of the 21th International Conference on Applications and Theory of Petri Nets*, volume 1825 of *LNCS*, pages 16–23, 2000.
- [16] M. de Oliveira Oliveira. Hasse diagram generators and Petri nets. *Fundamenta Informaticae*, 105(3):263–289, 2010.
- [17] M. de Oliveira Oliveira. Canonizable partial order generators. In *Proc. of the 6th International Conference on Language and Automata Theory and Applications (LATA 2012)*, volume 7183 of *LNCS*, pages 445–457, 2012.
- [18] V. Diekert. A partial trace semantics for petri nets. *Theoretical Computer Science*, 134(1):87–105, 1994.
- [19] M. Droste. Concurrent automata and domains. *International Journal of Foundations of Computer Science*, 3(4):389–418, 1992.
- [20] A. Ehrenfeucht and G. Rozenberg. Partial (set) 2-structures. Part I: Basic notions and the representation problem. *Acta Informatica*, 27(4):315–342, 1989.
- [21] J. Engelfriet and J. J. Vereijken. Context-free graph grammars and concatenation of graphs. *Acta Informatica*, 34, 1997.
- [22] J. Esparza, S. Römer, and W. Vogler. An improvement of mcmillan’s unfolding algorithm. *Formal Methods in System Design*, 20(3):285–310, 2002.
- [23] J. Fanchon and R. Morin. Pomset languages of finite step transition systems. In *Petri Nets*, volume 5606 of *LNCS*, pages 83–102, 2009.
- [24] H. Gaifman and V. R. Pratt. Partial order models of concurrency and the computation of functions. In *Proc. of the 2nd Symposium on Logic in Computer Science (LICS 1987)*, pages 72–85, 1987.
- [25] T. Gazagnaire, B. Genest, L. Hélouët, P. S. Thiagarajan, and S. Yang. Causal message sequence charts. In *Proc. of the 18th International Conference in Concurrency Theory (CONCUR 2007)*, volume 4703 of *LNCS*, pages 166–180. Springer, 2007.
- [26] B. Genest, A. Muscholl, H. Seidl, and M. Zeitoun. Infinite-state high-level mscs: Model-checking and realizability. *Journal of Computer and System Sciences*, 72(4):617–647, 2006.
- [27] D. Giammarresi and A. Restivo. Recognizable picture languages. *International Journal of Pattern Recognition and Artificial Intelligence*, 6(2-3):241–256, 1992.
- [28] D. Giammarresi and A. Restivo. Two-dimensional finite state recognizability. *Fundamenta Informaticae*, 25(3):399–422, 1996.

- [29] J. L. Gischer. The equational theory of pomsets. *Theoretical Computer Science*, 61:199–224, 1988.
- [30] U. Goltz and W. Reisig. Processes of place/transition-nets. In *Proc. of ICALP*, volume 154 of *LNCS*, pages 264–277, 1983.
- [31] J. Grabowski. On partial languages. *Fundamenta Informaticae*, 4(2):427, 1981.
- [32] J. Hayman and G. Winskel. The unfolding of general petri nets. In *FSTTCS*, volume 2 of *LIPIcs*, pages 223–234. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2008.
- [33] J. G. Henriksen, M. Mukund, K. N. Kumar, M. A. Sohoni, and P. S. Thiagarajan. A theory of regular MSC languages. *Information and Computation*, 202(1):1–38, 2005.
- [34] P. Hoogers, H. Kleijn, and P. Thiagarajan. A trace semantics for Petri nets. *Information and Computation*, 117(1):98–114, 1995.
- [35] P. W. Hoogers, H. C. M. Kleijn, and P. S. Thiagarajan. An event structure semantics for general Petri nets. *Theoretical Computer Science*, 153(1–2):129–170, 1996.
- [36] J.-F. Husson and R. Morin. On recognizable stable trace languages. In *In Proc. of the 3rd International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2000)*, volume 1784 of *LNCS*, pages 177–191, 2000.
- [37] O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25(1):116–133, 1978.
- [38] L. J. Jagadeesan and R. Jagadeesan. Causality and true concurrency: A data-flow analysis of the pi-calculus. In *Proc. of the International Conference on Algebraic Methodology and Software Technology*, volume 936 of *LNCS*, pages 277–291. Springer, 1995.
- [39] L. Jategaonkar and A. R. Meyer. Deciding true concurrency equivalences on safe, finite nets. *Theoretical Computer Science*, 154(1):107–143, 1996.
- [40] G. Juhás, R. Lorenz, and J. Desel. Can I execute my scenario in your net? In *Proc. of the 26th International Conference in Applications and Theory of Petri Nets*, volume 3536 of *LNCS*, pages 289–308, 2005.
- [41] O. Kupferman, Y. Lustig, M. Y. Vardi, and M. Yannakakis. Temporal synthesis for bounded systems and environments. In *Proc. of the 28th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2011)*, volume 9 of *LIPIcs*, pages 615–626, 2011.
- [42] D. Kuske and R. Morin. Pomsets for local trace languages. *Journal of Automata, Languages and Combinatorics*, 7(2):187–224, 2002.
- [43] R. Langerak, E. Brinksma, and J.-P. Katoen. Causal ambiguity and partial orders in event structures. In A. Mazurkiewicz and J. Winkowski, editors, *Proc. of the 8th International Conference on Concurrency Theory (Concur 1997)*, volume 1243 of *LNCS*, pages 317–331, Warsaw, Poland, 1997. Springer-Verlag.

- [44] K. Lodaya and P. Weil. Series-parallel languages and the bounded-width property. *Theoretical Computer Science*, 237(1-2):347–380, 2000.
- [45] S. Mauser. *Synthese von Petrinetzen aus halbgeordneten Abläufen*. PhD thesis, der Fakultät für Mathematik und Informatik der FernUniversität in Hagen, September 2010. (<http://deposit.fernuni-hagen.de/2768>).
- [46] A. W. Mazurkiewicz. Trace theory. In *Proc. of the 7th International Conference on Applications and Theory of Petri Nets (Petri Nets 1986)*, volume 255 of *LNCS*, pages 279–324, 1986.
- [47] K. L. McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *In Proc. of the Fourth International Workshop on Computer Aided Verification*, volume 663 of *LNCS*, pages 164–177, 1992.
- [48] U. Montanari and M. Pistore. Minimal transition systems for history-preserving bisimulation. In *Proc. of the 14th Annual Symposium on Theoretical Aspects of Computer Science (STACS 1997)*, volume 1200 of *LNCS*, pages 413–425, 1997.
- [49] R. Morin. On regular message sequence chart languages and relationships to mazurkiewicz trace theory. In *Proc. of the 4th International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2001)*, volume 2030 of *LNCS*, pages 332–346, 2001.
- [50] A. Muscholl and D. Peled. Message sequence graphs and decision problems on mazurkiewicz traces. In *Proc. of the 24th International Symposium on Mathematical Foundations of Computer Science (MFCS99)*, volume 1672 of *LNCS*, pages 81–91, 1999.
- [51] A. Muscholl, D. Peled, and Z. Su. Deciding properties for message sequence charts. In *Proc. of the First International Conference on Foundations of Software Science and Computation Structure (FoSSaCS)*, volume 1378 of *LNCS*, pages 226–242, 1998.
- [52] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. of the Sixteenth ACM Symposium on the Principles of Programming Languages*, pages 179–190, 1989.
- [53] J. Sakarovitch. The ”last” decision problem for rational trace languages. In I. Simon, editor, *Proc. of 1st Latin American Symposium on Theoretical Informatics (LATIN 1992)*, volume 583 of *LNCS*, pages 460–473, Berlin, Germany, 1992. Springer.
- [54] W. Thomas. Finite-state recognizability of graph properties. *Theorie des Automates et Applications*, 172:147159, 1992.
- [55] W. Vogler. *Modular Construction and Partial Order Semantics of Petri Nets*, volume 625 of *LNCS*. Springer, 1992.